ON THE SOLUTION OF TRAVELING SALESMAN PROBLEMS UNDER CONDITIONS OF SPARSENESS

A THESIS

Presented to

The Faculty of the Division of Graduate Studies and Research

by

Norman Jon Bau

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Operations Research

in the School of Industrial and Systems Engineering

Georgia Institute of Technology

December, 1976

ON THE SOLUTION OF TRAVELING SALESMAN PROBLEMS
UNDER CONDITIONS OF SPARSENESS

Approved:

Robert G. Parker, Chairman

C. M. Shetty

John J. Jarvis

Date Approved by Chairman: _____

ACKNOWLEDGMENTS

This thesis is dedicated to my mother, who inspired its completion and strengthened by determination throughout the research.

The guidance and direction for this work was provided by Dr. R. G. Parker. Despite his busy schedule, Dr. Parker always found time to review aspects of the work and make helpful comments and suggestions. His competency and interest was a motivating factor for me.

Drs. M. C. Shetty and J. J. Jarvis, the members of my reading committee, made numerous comments and suggestions, all of which served to broaden my perspective of the problem. Their contributions have been most valuable.

The final version of this thesis was typed by Mrs. Nancy Price.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF ILLUSTRATIONS

SUMMARY

The traveling salesman problem on a sparse graph has been considered, and the evidence gained in this thesis indicates that the intuitive notion of increasing computational effort relative to increasing density is not, in fact, substantiated. This result is indicated on three distinct classes of problems: non-symmetric, symmetric non-Euclidean, and symmetric-Euclidean. A specific subtour elimination algorithm is used as a vehicle in achieving this result. The procedure is detailed in algorithmic form and demonstrated computationally.

Two heuristic algorithms are presented, one of which applies the sparse graph results toward the solution of the complete graph problem. The other presents a modified branching rule in the spirit of the aforementioned subtour elimination algorithm. Computational results of both heuristics indicate a substantial reduction in computational effort with a very small deviation from optimality. These encouraging results are obtained through a consideration of all three previously described major problem classes.

CHAPTER I

INTRODUCTION

The application of Operations Research and in particular, the tools of optimization, has been manifest in numerous real world settings. Included, are problems in areas such as production, transportation, the environment, and even in health care delivery systems. The entire realm of problem modelling and analysis using techniques such as linear and integer programming, dynamic programming, implicit enumeration, and simulation have been influential in treating important problems, many of which are especially critical in times of scarse resources. Among problems of substantial concern are those requiring discrete solutions. Typical of such problems are those arising in job-shop scheduling, vehicle routing, and transportation planning. While some of these problems give way to special structures which facilitate their solution, others remain for the most part, unsolved. This thesis pertains to one such problem.

Discrete optimization encompasses problems which are among the most difficult to treat. Due to the combinatorial nature of the problems, their solutions command an exponential time growth rate as problem size increases. The result is that problems of even moderate, real world size cannot be treated in an exact mode.

The traveling salesman problem is one of the most celebrated in discrete optimization. In its classic version it is required that a tour through a set of n points be constructed such that the total tour

length is minimized. Further, it is required that each point be visited on the tour exactly once. One can easily represent the problem by a graph. When the cost of going from one node in the graph to another distinct node is finite, for all pairs of distinct nodes, a complete graph is said to exist. In real world applications however, it may not be possible to travel between all nodes with finite cost (e.g. some arcs may not even exist). A sparse graph or sparse problem is said to exist in this case and is an important problem in its own right. An appealing solution method for the sparse problem is to apply an algorithm for the classic traveling salesman problem, which we usually take to imply the existence of a complete graph. The results of this thesis would suggest that the simple application of a traditional traveling salesman algorithm can result in substantial computational variation under conditions of sparseness.

Literature Review

In this section we review procedures which have been developed to find the minimal, or near minimal cost circuit or tour in moderate sized traveling salesman problems. Generally, the organization of the review of literature will be with regard to technique type: exact versus heuristic. The exact procedures are those which can guarantee optimality and will be presented first. Heuristic procedures are approximate and cannot guarantee optimality at termination. The computational effort of heuristics is generally much lower than exact procedures which, of course, provides the basis for their appeal.

Exact Procedures

One of the most recent contributions to the attack on the traveling salesman problem is the work of Held and Karp.[17] Defining a minimal 1-tree, which is closely akin to a minimum spanning tree, an iterative method was developed which converges to an optimal solution. The procedure employs a branch and bound strategy in certain cases. To construct a 1-tree, one initially constructs a minimum spanning tree on the node set {2, 3, ..., n} where n is the number of "cities" in the problem. Node 1 is next adjoined to the tree via the two least cost arcs. The primal-dual method of Held and Karp ascends through successive 1-trees until either a tour is determined or no further ascent is possible. Termination with a tour is sufficient to provide optimality. When a branch is made, the usual process of branch-and-bound is continued and 1-trees are again generated. Each 1-tree is a lower bound on the optimal solution. If any 1-tree on a branch has a cost which is equal to or exceeds the minimal cost tour yet found, that branch is terminated. The largest problem yet solved with the Held-Karp algorithm is on the order of 60 cities.[18] A major weakness of the algorithm is its ability to solve only symmetric problems (relative to (4)).

A circuit elimination algorithm was investigated by Bellmore and Malone.[4] The procedure is essentially a branch and bound algorithm where successive assignment solutions are obtained within a branching scheme until all branches are fathomed and the lowest cost tour is determined. More explicitly, the initial cost matrix is solved as an assignment problem. If the assignment solution forms a Hamiltonian circuit or tour, then this is the optimal solution to the problem. If

not, then there must be subtours, or loops, formed in the assignment solution, and an exhaustive partition is made. Branching is performed such that the subtour of smallest cardinality is considered. In each branch, a different node of the subtour is selected and all arcs from this node to other nodes within the subtour are prohibited. New assignment problems are then solved on each branch until all branches are fathomed and the least cost tour is determined.

Eastman[12] and Little, et. al.[24] also developed branch and bound techniques. Eastman's work has since been extended by Shapiro[31] to cover symmetric problems. In particular, Eastman and Shapiro's algorithms employ the circuit elimination notion; whereas, Little's is a tour building algorithm. The Little algorithm branches on links or arcs which are generated in a least cost manner. A binary partition is made where in one case the link is fixed in the tour, and excluded from consideration or assigned infinite cost in the other. At each partition a lower bound is generated. When all lower bounds equal or exceed the lowest cost tour yet found, the algorithm terminates with the optimal solution. It can be remarked that Little's algorithm is something of a landmark in the development of solution techniques for the traveling salesman problem, as well as with reference to the theory of branch and bound in general.

The Eastman algorithm also solves a sequence of assignment problems. Each assignment solution is a lower bound to the cost of a tour on that branch. If the lower bound at a given assignment is not less than the least cost tour yet found, no branches are made and the algorithm returns to another part of the tree (backtracking). When subtours exist

in an assignment solution which has cost less than the least cost tour yet found, Eastman determines the least cardinality subtour. Let i_1, i_2, ..., i_k be the nodes representing the subtour. The algorithm forms k branches, creating k new assignment subproblems. In subproblem j the arc (i_j, i_{j+1}) is prohibited or assigned infinite cost for $1 \leq j \leq k-1$. Similarly, subproblem k prohibits the arc (i_k, i_1). The algorithm converges to an optimal solution upon termination, which occurs when all assignment solutions with subtours have costs not less than the least cost tour yet found. Computational experience indicates that the Eastman algorithm (the branching strategy) is generally inferior to that of Bellmore-Malone.

A major weakness of the subtour elimination procedures arises when symmetric problems are treated. In the case of symmetric problems, there are a great number of 2-city subtours formed, and this greatly limits the size of problems which may be solved. Edmonds[13] considers a redefinition of the variables in the linear program to implicitly eliminate 2-city subtours. A mutually exclusive branching scheme is developed by Bellmore and Malone which reduces somewhat the problems associated with subtour generation in the symmetric problem, yet eliminates no tours.

Svestka and Huckfeldt[32] considered the M-salesman traveling salesman problem. They develop an initial tour generation technique, and extend the Bellmore-Malone branching technique to this problem. The M-salesman algorithm when applied to the single salesman problem matches the results of Bellmore and Malone. The significant contribution is the initial tour generation algorithm. The M-salesman algorithm solves a

sequence of assignment problems. If the first assignment solution is a tour, then it is optimal. Otherwise, subtours exist which must be broken. The initial tour generator finds the least cost merging of two subtours into one by trading two arcs not in the subtours for two arcs contained in the subtours. This reduces the number of subtours by one, and is re-applied until a tour exists. This heuristic procedure provides a strong upper bound on the cost of an optimal tour and results in the early termination of many branches in the tree. Its major weakness is its inability to handle moderately sized symmetric problems.

Bellman[2], Gonzalez[15], and Held and Karp[16] all developed dynamic programming approaches to the problem. By defining $f_{k-1}(i_m|i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_{k-1})$ to be the shortest path from node 1 to node i_m which passes through nodes $i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_{k-1}$ and by recursively using the equation:

$$f_k(j|i_1, \ldots, i_{k-1}) = \min_{m=1,\ldots,k-1}[f_{k-1}(i_m| i_1, \ldots, i_{m-1}, i_{m+1}, \ldots, i_{k-1}) + C_{i_m j}]$$

an optimal solution can be obtained. The serious drawback which prevents the use of dynamic programming to solve moderate sized problems is that all available core storage is used up in attempting to store the large number of variables required at each level.

Integer programming was attempted as a solution technique by Dantzig, Fulkerson, and Johnson[10] in 1954, Miller, Tucker, and Zemlin[26] in 1960 and Martin[25] in 1966. Except for a 42-city problem which both Dantzig and Martin were able to solve, the enormous number of subtour constraints and the 0-1 variables make it impossible to solve moderate

sized problems using this technique.

Heuristic Procedures

The previous algorithms are exact or optimal procedures. Other algorithms exist which are heuristic or approximate and are useful for computational expediency. The nearest unvisited city algorithm is one of the oldest such procedures of this type. Here, a starting node is chosen and the nearest neighbor is determined; this link is fixed. Other links are prohibited so as to prevent subtour generation. The nearest neighbor is again determined and the algorithm proceeds in this manner until a tour is formed. There are several variations of the nearest unvisited city algorithm. One particularly well known one is given by Karg and Thompson.[20]

Ashour, Vega, and Parker[1] have developed a heuristic algorithm based upon the concept of regrets. The initial cost matrix is reduced, to generate zeroes or favorable arcs. Developing bounds on the cost of any tour if these zeroes are prohibited, the largest such bound is determined and the appropriate zero link is fixed so as to avoid a large increase in the bound. Appropriate links are prohibited so as to avoid subtours. The algorithm proceeds in this manner until a tour is generated. The difficulty with this and other heuristics is that the quality of solution is dependent upon the starting conditions and, of course, optimality at termination cannot be guaranteed.

Reiter and Sherman[29] started with an initial tour, say, (1, 2,..., n, 1) and attempted to relocate node 1 elsewhere in the chain so that a lower cost resulted. Then node 2 would be relocated and so forth, until n iterations were carried out without any relocations. The first two

nodes, say, (i_1, i_2) would next be relocated if possible, in the chain to reduce the cost of a tour. Relocation of the next two nodes would then be attempted. This process would continue until no improvement results after n applications. Then the relocation of one node at a time would be again attempted. Next, three node relocation would be attempted followed by single node relocation, and so forth.

Lin[22] employed a so-called λ-opt concept to the problem. A tour is λ-opt if no superior tour can be obtained by replacing any λ arcs, and if the tour is $(\lambda-1)$-opt. Lin and Kernighan[23] use the following fact to extend the λ-opt concept: if a sequence of numbers has a positive sum, there is a cyclic permutation of these numbers such that every partial sum is positive. The parameter λ is not fixed in advance, but links are broken and formed in the spirit of the λ-opt concept so as to maximize the reduction in cost. Christofides and Eilon[7] extend the λ-opt concept by developing a formula which determines the maximum reduction in cost possible by a recombination of arcs in a certain class. This places constraints on the arcs which may be recombined, and computational effort is decreased as fewer alternative recombinations need be considered. This is combined with the fact that all λ-opt tours can be formed from a finite number of recombinations of 2-opt tours to solve larger problems than can be solved with the λ-opt method.

Holmes and Parker[19] modified the savings approach of Clark and Wright as applied to the delivery problem, of which, the traveling salesman problem is a special case. Perez and Parker[27] applied the work of Holmes to the traveling salesman problem. By picking a node as a central "depot", a process is considered where the salesman travels to the first

node, returns to the depot, travels to the next node and returns to the depot and so on until all cities are visited. They define the savings gained by going directly from each city to another, bypassing the depot, and fix links such that the maximum savings is gained as each link is fixed. Arcs are prohibited and savings adjusted so as to prohibit subtours. In the next phase, the first link in the tour is prohibited and the problem re-solved. If a lower cost tour results, this solution is retained as the incumbent and the prohibited arc remains so permanently. Then the first link of the new incumbent solution is prohibited and the problem re-solved. If the re-solved problem has a cost higher than the initial, the prohibited link is restored to its previous value, and the next link chosen is prohibited. The algorithm proceeds in this manner until termination, which occurs when all links in a proposed solution are prohibited and no lower cost tours result, or when a specified number of links have been prohibited without resulting in an improved solution. A weakness of this algorithm is that too much computational effort is required to solve n problems using each city as the depot. However, the authors suggest encouraging results when the city located closest to the centroid is used as the depot. In its current form this restricts the algorithm to the case of the Euclidean problem.

In this review of the literature the major and/or recent contributions for treating the traveling salesman problem have been highlighted. Obviously, the importance as well as the appeal of the problem has created substantial literature on methods for attacking the traveling salesman problem. A more complete list of references can be found in the excellent survey paper by Bellmore and Nemhauser[3] as well as in Christofides[8] and Turner, et al.[34].

Outline of the Thesis

The organization of this thesis proceeds such that Chapter II is concerned with the solution of the traveling salesman problem defined on the sparse graph. Computational experience related to this topic is presented in Chapter III. In Chapter IV two heuristic procedures are suggested. One procedure capitalizes on the conditions of sparseness and the other heuristic procedure is a k-1 node branching scheme in the spirit of the Bellmore-Malone algorithm. Both techniques are presented with computational experience. The final chapter summarizes the results of this thesis and suggests encouraging extensions of the work.

CHAPTER II

ON TREATING PROBLEMS WITH SPARSE GRAPHS

This chapter is concerned with several areas, the most important of which is the effect of sparcity on the traveling salesman problem. The complete graph problem will be defined as a mathematical model, and some sufficiency conditions for the existence (or lack thereof) of a tour will be considered. The Bellmore-Malone algorithm will be demonstrated, and the effect of sparcity will be considered through an example.

Some Conditions for the Existence of Tours in a Graph and the Bellmore-Malone Algorithm

Two general classes of graphs are generally considered: symmetric and non-symmetric. Symmetric problems are those in which the cost of travel between a pair of distinct nodes is independent of the direction of travel, for all pairs of distinct nodes. If c_{ij} is defined as the cost of travel between nodes i and j then $c_{ij} = c_{ji}$ for a symmetric problem. The degree of a node i, denoted deg(i), in a symmetric problem is defined as the number of connecting arcs from node i to all other nodes. A weak sufficiency condition for the existence of a tour in a symmetric graph is the theorem of Dirac[11], stated in the following proposition.

Proposition 1. Consider a symmetric graph with n nodes labelled 1, 2, ..., n. If $\deg(i) \geq \frac{n}{2}$ for i=1, 2, ..., n, then there exists a tour through the n nodes.

A necessary condition for the existence of a tour is given by the following proposition.

Proposition 2. If a tour exists through the n nodes labelled 1, 2, ..., n on a symmetric graph, then deg(i) $\geq$ 2 for i=1, 2, ..., n.

This is obvious since if deg(i) = 1 a pendant node is formed such that a path can arrive at a node but cannot depart without traversing the same arc. If deg(i) = 0 the graph is not connected. The following proposition details a necessary condition for the existence of a tour on a non-symmetric graph.

Proposition 3. If a tour exists on a non-symmetric graph then there is at least one connecting arc directed out of each node and at least one connecting arc directed into each node. This follows by the same rationalle as given for the second proposition.

As in many real world applications a mathematical model is necessary to effect a solution or develop properties exhibited by the problem. In formulating a mathematical model for the traveling salesman problem, two properties are considered which characterize the problem: (1) each city must be visited exactly once, and (2) there can be no subtours or loops. Property (2) infers that the only feasible closed path passes through all the nodes.

Let c_{ij} be the cost of travel between cities i and j. Let x_{ij} be 1 if the path between cities i and j is chosen, and 0 otherwise. Further, let n denote the number of cities in the problem. A subtour or loop exists when there k arcs, k < n which form a tour of k cities. For example, in a 6-city problem the arcs (1, 2, 3, 6, 1) and (4, 5, 4) form 4 and 2 city subtours respectively. Simply stated, the objective then is to visit each city exactly once in a least cost manner while precluding the occurrence of subtours. Consider the following formulation (P1) and

let M be the number of subtours possible in an n-city graph.

$$\text{P1:} \qquad \text{Minimize} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\text{subject to:} \quad \sum_{j=1}^{n} x_{ij} = 1 \quad \text{for } i=1, 2, \ldots, n \tag{1}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \text{for } j=1, 2, \ldots, n \tag{2}$$

$$\sum_{(i,j) \in C_L} x_{ij} \leq |C_L| - 1 \quad L=1, 2, \ldots, M \tag{3}$$

$$x_{ij} \in [0,1] \qquad i, j=1, 2, \ldots, n$$

where C_L is the set of arcs (i,j) which form loop L for L=1, 2, ..., M.

As previously stated, the two major classes of graphs are symmetric and non-symmetric. Within symmetric graphs there are two major divisions, Euclidean and non-Euclidean. Euclidean graphs are distinguished from non-Euclidean ones in that the arcs satisfy the triangle inequality. Distances taken from a map consitute a Euclidean graph for example. Of the three problem types considered, non-symmetric, symmetric-Euclidean, and symmetric non-Euclidean, the Bellmore-Malone[4] algorithm is probably one of the best exact procedures developed to date. Its ability to solve all three types of problems makes it superior in this application. Consequently, this algorithm will serve as a vehicle in the investigation of the effect of sparcity on these three problem types. Following, we describe the Bellmore-Malone procedure in a step by step fashion.

Step 1: Define a variable Z to represent the lowest cost tour yet found. Initially, set Z at some sufficiently large value, M.

Step 2: Solve the assignment problem resulting from the initial cost matrix.

Step 3: If a tour results set Z equal to the cost of the tour and proceed to step 11. Otherwise, proceed to step 4.

Step 4: Determine the minimal cost assignment solution node and the associated assignment solution. If a tie exists, it is randomly broken.

Step 5: If this minimal cost node has value greater than or equal to Z proceed to step 11.

Step 6: Determine the least cardinality (smallest) subtour in this assignment solution. If a tie exists, it is randomly broken. Let this cardinality be k.

Step 7: Create k new subproblems. Let the k nodes in the subtour be N_1, N_2, ..., N_k. Subproblem i is formulated by changing the cost of the arc (N_i, N_r) to M in the cost matrix for r=1, 2,..., i-1, i+1, ..., k for i=1, 2, ..., k. The cost matrix is that of the optimal assignment solution considered, and M is an arbitrarily large number.

Step 8: Assign to each of the k new subproblems a cost equal to the optimal solution of the assignment problem from which they were generated.

Step 9: Determine the minimal cost node and associated assignment problem. If a tie exists, it is randomly broken. Solve this assignment problem.

Step 10: If a tour results, set Z equal to the minimum of the current value of Z and the cost of the tour. Otherwise, the value of Z remains unchanged. Return to step 4.

Step 11: The optimal solution is given by the current value of Z. The algorithm terminates at this point.

The Bellmore-Malone algorithm is a subtour elimination technique which solves successive assignment problems within the general framework of branch and bound. The smallest cardinality subtour is chosen for branching so as to minimize the number of assignment problems solved.

The Effect of Sparcity

Following is an example of the Bellmore-Malone algorithm on a complete graph problem. The initial cost matrix is given in Table 2.1. This matrix represents assignment problem A1. The Hungarian algorithm is employed to solve all assignment problems. Upon application to problem A1 the optimal assignment matrix is found and appears in Table 2.2. The assignment solution is (1, 2, 6, 5, 7, 4, 3, 8, 1) which is also a tour with cost 55. Obviously, this tour is an optimal one since it corresponds to the initial assignment solution.

Consider now the sparse graph whose costs are given in Table 2.3. (See Figure 2.1 for an example of a sparse graph.) This graph has a density of .84 and was created by randomly prohibiting arcs from the previously considered complete graph. The optimal assignment matrix is given in Table 2.4. The initial cost matrix and assignment problem for this sparse graph is denoted as problem A1. The optimal assignment solution has cost 56 and two 4-city subtours result: (1, 2, 6, 8, 1) and (3, 5, 7, 4, 3). There is a tie for the least cardinality subtour and (1, 2, 6, 8, 1) is arbitrarily chosen for subtour elimination. Four subproblems are created (see Figure 2.2), A2, A3, A4, and A5. Subproblem A2 is created by prohibiting the arcs (1,2), (1,6) and (1,8) in the optimal assignment matrix given in Table 2.4. Problem A3 is created by prohibiting the arcs (2,6), (2,8), and (2,1) in the matrix given in Table 2.4 The

TABLE 2.1. The Initial Cost Matrix for the Sample Problem, A1.

-	4	21	10	12	23	27	8
21	-	29	10	21	4	5	16
21	9	-	9	11	1	22	1
24	17	9	-	10	17	9	20
18	30	18	11	-	18	6	25
25	25	20	26	14	-	30	5
8	16	11	1	21	11	-	23
16	23	22	15	21	22	14	-

TABLE 2.2. The Optimal Assignment Matrix for Problem A1.

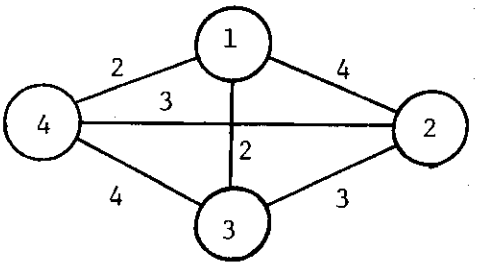
-	0	17	8	7	27	30	12
7	-	17	0	8	0	0	12
10	0	-	2	1	0	20	0
13	8	0	-	0	16	7	19
3	17	5	0	-	13	0	20
10	12	7	15	0	-	24	0
3	13	8	0	17	16	-	28
0	9	8	3	6	16	7	-

TABLE 2.3. The Initial Cost Matrix for a Sparse Graph with Approximate Density .84

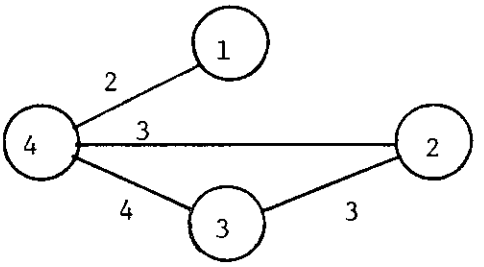
-	4	21	10	12	23	27	8
21	-	29	10	-	4	5	16
-	9	-	9	11	-	22	1
24	17	9	-	10	17	9	20
18	30	18	11	-	18	6	25
25	-	-	26	-	-	30	5
8	-	-	1	21	11	-	23
16	23	-	15	21	22	14	-

TABLE 2.4. The Optimal Assignment Matrix for Problem A1.

-	0	16	6	6	18	22	12
15	-	25	7	-	0	1	21
-	0	-	0	0	-	12	0
13	9	0	-	0	8	0	20
10	25	12	6	-	12	0	28
9	-	-	13	-	-	16	0
4	-	-	0	18	9	-	30
0	10	-	2	6	8	0	-



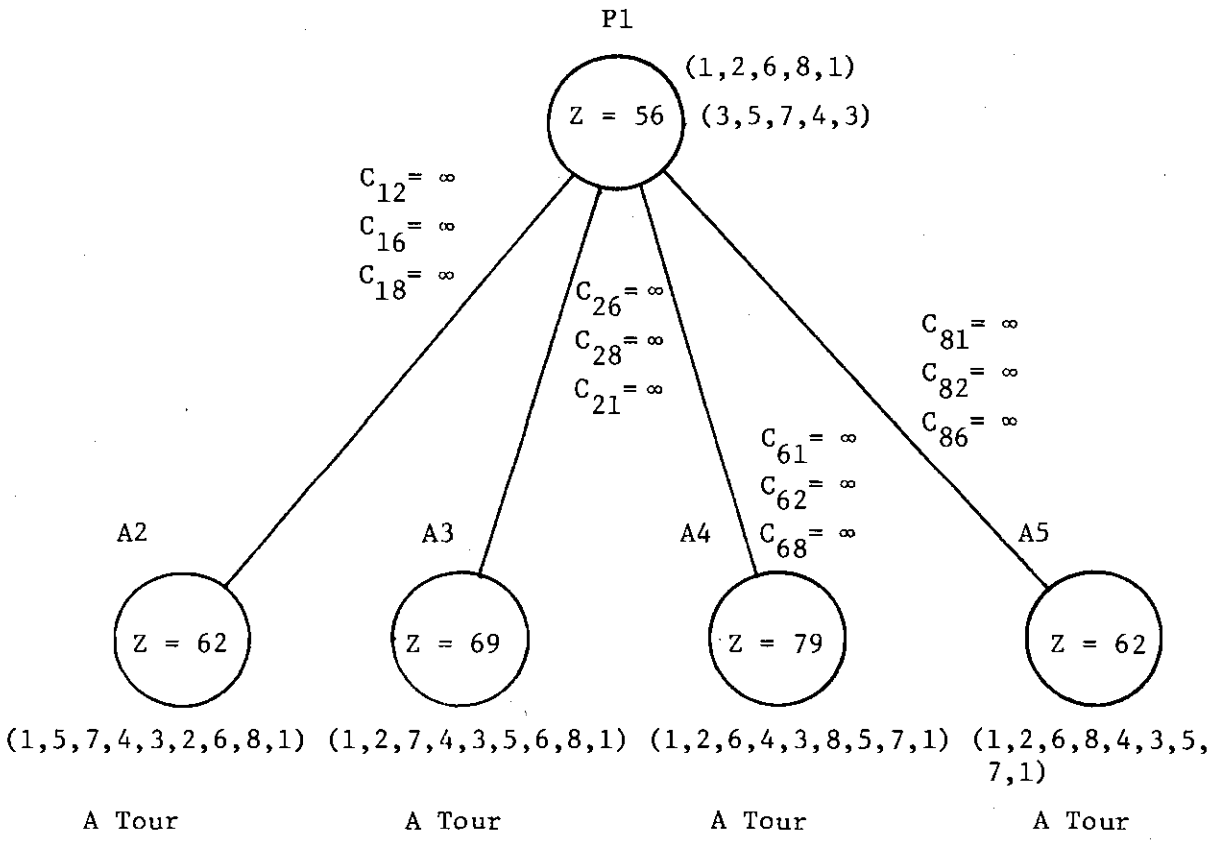
(a) A Complete 4-City Graph.



Approximate Density: .67

(b) A Sparse 4-City Graph.

Figure 2.1 Examples of Complete and Sparse Graphs.



Initial Cost Matrix Given in Table 2.3 with Density .84.

Figure 2.2 Application of the Bellmore-Malone Algorithm to the Sparse Problem.

other subproblems are similarly generated. The initial cost matrix for subproblem A2 is given in Table 2.5. The optimal assignment matrix for subproblem A2 is given in Table 2.6 and has cost 62. The solution is a tour, (1, 5, 7, 4, 3, 2, 6, 8, 1) and this becomes the lowest cost tour yet found.

Subproblems A3, A4, and A5 are next solved since their lower bound is 56, which is not greater than or equal to the lowest cost tour yet found (see Figure 2.2). Problems A3 and A4 have optimal costs of 69 and 79 respectively, and both are tours. Problem A5 has an optimal cost of 62 and it also yields a tour. Hence, the solution to the problem is an optimal tour of cost 62. The computation is summarized in Figure 2.2.

Consider next the sparse matrix given in Table 2.7. It, too, was formed from the complete graph matrix by randomly prohibiting arcs and has a density of just under .60. Denoting the assignment problem with this cost matrix, problem A1, it is found to have an assignment solution of cost 84 (Table 2.8). Furthermore, the assignment solution (1, 2, 6, 8, 5, 4, 3, 7, 1) forms a tour and necessarily becomes the optimal solution to the problem.

One might suspect that the computational effort required to determine optimal tours in graphs with decreasing density will decrease. This is based upon the notion that since there are fewer arcs, there exists fewer tours and therefore, less searching need be performed to locate an optimal solution. The example problem suggests that this is not the case. The complete graph and .60 density graph problems required approximately the same computational effort while the graph of .84 density required about 5 times more computational effort.

TABLE 2.5. The Initial Cost Matrix for Subproblem A2.

-	-	16	6	6	-	22	-
15	-	25	7	-	0	1	21
-	0	-	0	0	-	12	0
13	9	0	-	0	8	0	20
10	25	12	6	-	12	0	28
9	-	-	13	-	-	16	0
4	-	-	0	18	9	-	30
0	10	-	2	6	8	0	-

TABLE 2.6. The Optimal Assignment Matrix for Subproblem A2.

-	-	10	0	0	-	16	-
15	-	25	7	-	0	1	21
-	0	-	0	0	-	12	0
13	9	0	-	0	8	0	20
10	25	12	6	-	12	0	28
9	-	-	13	-	-	16	0
4	-	-	0	18	9	-	30
0	10	-	2	6	8	0	-

TABLE 2.7. The Initial Cost Matrix for a Sparse Graph with Density Approximately .60.

-	4	21	10	12	23	27	8
21	-	29	10	-	4	-	16
-	9	-	9	-	-	22	1
24	17	9	-	10	-	9	20
18	30	-	11	-	18	-	-
-	-	-	-	-	-	-	5
8	-	-	-	-	11	-	23
16	-	-	-	21	22	-	-

TABLE 2.8. The Optimal Assignment Matrix for Problem A1.

-	0	4	6	0	13	10	12
20	-	18	12	-	0	-	26
-	0	-	0	-	-	0	0
25	21	0	-	0	-	0	32
4	19	-	0	-	1	-	-
-	-	-	-	-	-	-	0
0	-	-	-	-	0	-	26
0	-	-	-	0	3	-	-

In the following chapter the effect of sparcity on computational effort is reported for all three types of graphs: non-symmetric, symmetric-Euclidean, and symmetric non-Euclidean. In addition, some issues arising from the effects of sparcity are addressed in order to better understand how sparcity is manifest in a problem solution.

CHAPTER III

THE EFFECT OF SPARCITY

This chapter is organized into three sections. In the first section the effect of sparcity on the non-symmetric problem will be considered. The second and third sections treat the effect of sparcity on symmetric non-Euclidean and symmetric-Euclidean problems respectively. The effect on total computational effort of various degrees of sparcity will be given particular importance in the coverage. Insight into the problem will be pursued by addressing both the number of branches generated and the computational effort required to pursue each branch. All computational experience is based upon the use of the Bellmore-Malone algorithm.

Non-Symmetric Problems

Initially a random non-symmetric problem on n nodes was generated from a discrete uniform distribution on the interval [1,1000). All costs were integers contained in this interval. Randomly chosen arcs were next prohibited in order to achieve varying degrees of density. The same random number seed was used in the generation of prohibited arcs at each density level. If S_1 and S_2 represent the set of prohibited arcs at density levels d_1 and d_2 respectively and if $d_1 \leq d_2$, then $S_2 \subseteq S_1$. Further, if Z_1 and Z_2 represent the optimal solutions to the graphs of density d_1 and d_2 respectively, then $Z_1 \geq Z_2$. This must be since every finite cost arc in the graph of density d_1 is contained in the graph of

density d_2. Finally, in the generation of a sparse non-symmetric graph, no arc was prohibited if it was the only arc directed into its terminal node or if it was the only arc directed out of its departure node. This was implemented in the spirit of Proposition 3 (Chapter 2) so as to formulate the maximum number of sparse graphs which contained a finite cost tour.

In any complete graph with n nodes, the total number of finite cost arcs is n(n-1). If there are k finite cost arcs in a sparse graph, its density is defined as $\frac{k}{n(n-1)}$. Graphs were generated for non-symmetric problems with 6 through 10 nodes, 15 nodes, and 20 nodes. Within each of these problems sparse graphs were generated with densities from .2 through 1.0 in increments of .1.

The number of CPU (central processor units) seconds required, the optimal cost, the number of assignment problems solved, and the CPU time per assignment problem were determined for all problems and density levels. All problems at each density level were replicated with different random numbers from 1 to 20 times depending upon computational effort required. The means are calculated and presented in Tables 3.1 through 3.8.

Symmetric, Non-Euclidean Problems

To investigate the manifestations of sparseness on symmetric, non-Euclidean problems, an approach similar to that of the previous section was taken. In fact, except for two modifications, the current approach is identical to that in the earlier section. A random, symmetric problem from the uniform distribution on the interval [1,1000) was generated,

TABLE 3.1. Computational Effort Required on 6-City, Non-Symmetric Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Solved
.2	7449.65	.161	3.50	.044
.3	2339.75	.095	2.15	.043
.4	2284.45	.102	2.25	.043
.5	2083.85	.147	3.20	.043
.6	1939.10	.141	3.15	.043
.7	1868.60	.133	2.95	.044
.8	1731.25	.130	2.95	.041
.9	1578.60	.113	2.60	.042
1.0	1439.20	.108	2.45	.042

Random numbers generated from uniform distribution on the interval [1,1000).

20 replications.

Random number seed: 42137

TABLE 3.2. Computational Effort Required on 7-City, Non-Symmetric Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	2662.60	.205	3.35	.058
.3	2335.95	.227	3.75	.058
.4	2180.05	.127	2.05	.060
.5	2012.10	.193	3.20	.057
.6	1888.35	.239	4.00	.056
.7	1694.90	.252	4.15	.057
.8	1531.80	.212	3.55	.055
.9	1485.25	.161	2.70	.057
1.0	1441.35	.189	3.15	.059

Random numbers generated from uniform distribution on the interval [1,1000).

20 replications.

Random number seed: 53217

TABLE 3.3. Computational Effort Required on 8-City, Non-Symmetric Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problems Solved
.2	2693.80	.496	6.25	.083
.3	2371.05	.518	6.60	.079
.4	2191.70	.366	4.80	.075
.5	2090.25	.514	6.80	.074
.6	1975.95	.470	6.40	.072
.7	1870.50	.499	6.80	.071
.8	1770.60	.299	4.00	.077
.9	1676.95	.479	6.30	.078
1.0	1598.90	.512	6.80	.076

Random numbers generated from uniform distribution on the interval [1,1000).

20 replications.

Random number seed: 64231

TABLE 3.4. Computational Effort Required on 9-City, Non-Symmetric Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	2410.65	.812	7.70	.098
.3	2309.20	.567	5.25	.110
.4	2040.05	.513	4.65	.103
.5	1988.85	.667	6.15	.103
.6	1797.15	.687	6.45	.101
.7	1708.50	.831	7.85	.102
.8	1644.50	.649	6.30	.103
.9	1600.55	.726	6.85	.107
1.0	1526.65	.749	7.05	.106

Random numbers generated from uniform distribution on the interval [1,1000).

20 replications.

Random number seed: 44317

TABLE 3.5. Computational Effort Required on 10-City, Non-Symmetric Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.249	1.90	.134
.3	∞	.529	3.90	.148
.4	∞	1.996	15.30	.132
.5	∞	3.562	26.85	.134
.6	3377.20	4.729	35.50	.134
.7	2926.98	3.727	28.55	.131
.8	2565.63	4.182	32.30	.132
.9	2341.70	4.105	32.55	.128
1.0	2196.93	4.353	34.00	.130

Random numbers generated from uniform distribution on the interval [1,1000).

20 replications.

Random number seed: 33311

TABLE 3.6. Computational Effort Required on 15-City, Non-Symmetric Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	2628.80	4.646	13.59	.424
.3	2449.50	5.928	18.85	.326
.4	2282.85	5.540	18.70	.317
.5	2114.75	4.011	12.90	.337
.6	2053.10	5.083	16.30	.356
.7	1984.60	6.520	20.80	.365
.8	1788.95	6.594	21.35	.370
.9	1656.40	4.551	15.20	.372
1.0	1577.90	5.457	18.15	.340

Random numbers generated from uniform distribution

on the interval [1,1000).

20 replications.

Random number seed: 41411

TABLE 3.7. Computational Effort Required on 20-City, Non-Symmetric Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	2904.8	12.361	18.9	.759
.3	2665.0	9.923	14.2	.982
.4	2477.5	9.370	13.3	.781
.5	2358.5	14.853	23.8	.809
.6	2198.8	22.396	34.5	.654
.7	2122.0	15.840	24.8	.825
.8	2003.6	9.758	15.5	.899
.9	1949.4	17.832	27.6	.672
1.0	1889.5	24.213	38.5	.654

Random numbers generated from uniform distribution on the interval [1,1000).

10 replications.

Random number seed: 15791

TABLE 3.8. Computational Effort Required on 10-City, Non-Symmetric Problems with a Large Number of Replications.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	2652.62	1.163	8.647	.137
.3	2480.19	1.200	8.987	.140
.4	2322.43	1.284	9.713	.139
.5	2164.21	1.212	9.173	.139
.6	2000.47	1.266	9.600	.138
.7	1899.67	1.032	7.660	.139
.8	1802.55	.996	7.480	.136
.9	1698.38	1.018	7.667	.136
1.0	1620.85	1.170	8.873	.138

Random numbers generated from uniform distribution on the interval [1,1000).

150 replications.

in contrast to the non-symmetric problem generated previously. The other modification concerns the method of sparse graph formation. Since a symmetric cost problem suggests a graph which is undirected, it is desired to prohibit both an arc and its transpose element in the cost matrix. This has the effect of blocking travel on an arc in both directions. However, if either the arc or its transpose element represents the only finite cost link out of a node, neither the arc nor its transpose were prohibited.

Random, symmetric, non-Euclidean problems were generated with sizes ranging, as before, from 6 through 10 nodes, 15 nodes and 20 nodes. Within each of these problems sparse graphs of density .2 through 1.0 were generated in the manner described earlier.

The number of CPU seconds required, the optimal cost, the number of assignment problems solved, and the CPU time per assignment problem were determined for all density levels within each problem. Just as before, relative to computational effort required, 1 to 20 replications were performed and the means were calculated. All computational experience is presented in Tables 3.9 through 3.16.

Symmetric-Euclidean Problems

Symmetric-Euclidean and symmetric, non-Euclidean problems are identical save for one attribute. If arcs in the defining matrix are treated as distances, then the triangle inequality holds for all combinations of arcs in the symmetric-Euclidean graph. All statements and procedures of the previous section pertaining to the analysis of the effect of sparcity on symmetric, non-Euclidean problems are applicable to this section, except for one item. Rather than generating a random,

TABLE 3.9. Computational Effort Required on 6-City, Symmetric Problems

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs per Assignment Problem Solved
.2	∞	.083	2.0	.041
.3	∞	.083	2.0	.041
.4	∞	.101	2.3	.044
.5	∞	.167	3.4	.050
.6	∞	.337	7.0	.049
.7	∞	.292	6.1	.050
.8	7167	.353	7.3	.050
.9	1961	.328	6.8	.049
1.0	1764	.351	7.1	.049

Random numbers generated from uniform distribution

on the interval [1,1000).

20 replications.

Random number seed: 5533

TABLE 3.10. Computational Effort Required on 7-City, Symmetric Problems

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.057	1.0	.057
.3	∞	.083	1.3	.066
.4	∞	.134	2.1	.071
.5	∞	.290	4.6	.071
.6	∞	.435	6.9	.064
.7	∞	.544	8.7	.063
.8	2219.45	.632	10.0	.063
.9	2096.70	.591	9.7	.062
1.0	1900.55	.627	10.3	.062

Random numbers generated from uniform distribution

on the interval [1,1000).

20 replications.

Random number seed: 5511

TABLE 3.11. Computational Effort Required on 8-City, Symmetric Problems

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.087	1.4	.063
.3	∞	.139	1.8	.078
.4	∞	.447	5.8	.083
.5	∞	.849	11.0	.079
.6	∞	1.099	15.0	.074
.7	2769.35	1.240	16.1	.077
.8	2336.05	1.203	15.6	.077
.9	1987.70	1.112	14.6	.077
1.0	1834.00	1.212	16.4	.074

Random numbers generated from uniform distribution

on the interval [1,1000).

20 replications.

Random number seed: 91911

TABLE 3.12. Computational Effort Required on 9-City, Symmetric Problems

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.107	1.0	.107
.3	∞	.157	1.3	.125
.4	∞	.865	8.3	.117
.5	∞	1.389	12.9	.111
.6	3265.2	1.832	17.8	.105
.7	2658.0	2.092	19.5	.108
.8	2277.4	1.466	13.8	.107
.9	2012.0	1.858	17.6	.106
1.0	1808.5	2.139	20.5	.105

Random numbers generated from uniform distribution

on the interval [1,1000).

20 replications.

Random number seed: 66611

TABLE 3.13. Computational Effort Required on 10-City, Symmetric Problems

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.249	1.9	.134
.3	∞	.529	3.9	.148
.4	∞	1.996	15.3	.132
.5	8648.15	3.562	26.9	.134
.6	3113.35	4.729	35.5	.134
.7	2685.50	3.727	28.6	.131
.8	2314.30	4.182	32.3	.132
.9	2120.85	4.105	32.6	.128
1.0	1893.50	4.353	34.0	.130

Random numbers generated from uniform distribution

on the interval [1,1000).

20 Replications.

Random number seed: 33311

TABLE 3.14. Computational Effort Required on 15-City, Symmetric Problems

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.849	1.8	.533
.3	∞	14.134	44.6	.340
.4	∞	25.187	79.0	.312
.5	∞	38.695	125.9	.336
.6	∞	50.520	163.3	.327
.7	2793.4	24.733	78.8	.320
.8	2534.0	37.638	119.8	.314
.9	2327.1	39.599	127.9	.307
1.0	2057.3	23.765	78.9	.306

Random numbers generated from uniform distribution on the interval [1,1000).

10 replications.

Random number seed: 42137

TABLE 3.15. Computational Effort Required on 20-City, Symmetric Problems

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	5.664	7.0	.809
.3	6938.0	99.285	159.0	.624
.4	5517.0	94.760	143.0	.663
.5	5065.5	294.234	227.0	1.296
.6	3980.0	51.040	77.0	.663
.7	3810.0	276.244	426.0	.648
.8	3331.5	138.805	216.0	.643
.9	2950.5	298.138	466.0	.640
1.0	2714.0	128.752	202.0	.637

Random numbers generated from uniform distribution on the interval [1,1000).

2 replications.

Random number seed: 555331

TABLE 3.16. Computational Effort Required on 6-City, Symmetric Problems (Large Number of Replications)

Density	Optimal	CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.076	1.9	.040
.3	∞	.078	1.9	.041
.4	∞	.098	2.3	.042
.5	∞	.158	3.4	.046
.6	∞	.314	6.9	.046
.7	∞	.318	7.0	.046
.8	5406.44	.320	7.0	.046
.9	2001.67	.328	7.2	.046
1.0	1786.56	.335	7.5	.045

Random numbers generated from uniform distribution

on the interval [1,1000).

65 replications.

Random number seed: 91937

symmetric, non-Euclidean graph, the initial complete graph which is generated in this section is a sub-graph of the 57-city, Karg and Thompson Euclidean problem.[20] To generate replication k of an n-city, Euclidean problem, the first k-1 rows and columns of the 57x57 Euclidean matrix are first deleted. The first n rows and columns of this matrix form the desired Euclidean matrix. The sparse graphs are generated in exactly the same fashion as the previous sections.

Random, symmetric-Euclidean problems were generated for problem sizes ranging from 6 through 10 nodes, 15 nodes, and 20 nodes. Within each of these problems, sparse graphs of density .2 through 1.0 were generated. All computational experience is summarized in Tables 3.17 through 3.23.

Discussion of Computational Experience

In this section we evaluate the properties suggested by sparcity in the traveling salesman problem and which are indicated by the computational experience gained in the research.

Perhaps the most striking phenomenon of sparcity is the lack of any monotonocity of computational time as a function of density. As previously mentioned, the presence of fewer tours in the sparse graph suggests that it should require less computational effort to determine the optimum. However, the computational experience suggests that this notion is indeed false and that there are other properties of sparse graphs which are dominant in determining total computational effort required to solve the problem.

Consider the problem in which there exists relatively many tours which have a cost at or near the optimum of the problem. There will be

TABLE 3.17. Computational Effort Required on 6-City Symmetric-Euclidean Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.086	2.2	.039
.3	∞	.087	2.2	.039
.4	∞	.092	2.2	.042
.5	∞	.136	3.0	.046
.6	∞	.187	4.2	.045
.7	∞	.209	4.6	.046
.8	5534.0	.286	6.4	.045
.9	4882.9	.544	12.3	.045
1.0	4645.8	.900	20.7	.044

10 replications.

TABLE 3.18. Computational Effort Required on 7-City, Symmetric-Euclidean Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.052	1.0	.052
.3	∞	.073	1.2	.064
.4	∞	.093	1.4	.070
.5	∞	.168	2.6	.072
.6	∞	.487	8.2	.062
.7	5843.7	.606	10.4	.059
.8	5519.3	1.103	19.5	.058
.9	5263.9	1.619	28.2	.058
1.0	4977.3	2.694	47.6	.057

10 replications.

TABLE 3.19. Computational Effort Required on 8-City, Symmetric-Euclidean Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.098	1.6	.061
.3	∞	.151	2.2	.074
.4	∞	.460	6.4	.073
.5	∞	.554	7.5	.072
.6	∞	1.001	13.6	.079
.7	6548.4	1.291	18.1	.075
.8	5896.1	2.688	36.7	.073
.9	5653.5	3.707	51.0	.072
1.0	5336.8	8.093	109.4	.074

10 replications.

TABLE 3.20. Computational Effort Required on 9-City, Symmetric-Euclidean Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.114	1.0	.114
.3	∞	.165	1.4	.126
.4	∞	.437	3.8	.130
.5	∞	1.593	14.7	.115
.6	∞	2.862	27.7	.105
.7	6731.9	6.622	62.5	.109
.8	6224.4	6.381	61.3	.106
.9	5926.4	15.055	141.0	.108
1.0	5747.4	17.787	171.8	.107

10 replications

TABLE 3.21. Computational Effort Required on 10-City, Symmetric-Euclidean Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	.226	2.0	.118
.3	∞	.953	7.8	.132
.4	∞	1.779	13.2	.137
.5	∞	5.225	41.5	.123
.6	7403.7	4.425	34.8	.127
.7	7020.9	5.196	41.1	.131
.8	6740.1	13.670	111.4	.122
.9	6451.2	29.365	225.9	.129
1.0	6203.0	47.309	379.2	.126

1 replication.

TABLE 3.22. Computational Effort Required on 15-City, Symmetric-Euclidean Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	1.220	3	.407
.3	11872	34.613	112	.309
.4	10337	43.909	141	.311
.5	9635	52.193	159	.328
.6	9199	211.397	653	.324
.7	7549	23.314	75	.311
.8	5584	7.002	21	.333
.9	5584	30.673	106	.289
1.0	5584	195.669	667	.293

1 replication

TABLE 3.23. Computational Effort Required on 20-City, Symmetric-Euclidean Problems.

Density	Mean Optimal	Mean CPU Secs	Mean # of Assignment Problems Solved	Mean CPU Secs Per Assignment Problem Solved
.2	∞	8.484	15	.566
.3	12709	161.816	294	.550
.4	12300	252.696	427	.592
.5	10655	83.704	159	.526
.6	10037	101.211	157	.645
.7	10009	96.947	152	.638
.8	9487	190.419	301	.632
.9	9000	576.000	914	.630
1.0	8977	222.553	377	.590

1 replication

TABLE 3.24. Comparison of Selected Sparse and Complete Graph Problems.

Table Number Referenced	Percent Above Optimality	Percent Decrease in Computational Effort
2	3	15
3	5	6
6	5	17
7	3	26
8	5	13
17	5	40
18	6	40
19	6	54
20	3	15

The table number refers to the computational experience in the previous sections and the sparse graphs are those with density .9.

many assignment problems solved in the application of the Bellmore-Malone procedure and the lower bounds resulting at each assignment solution will increase rather slowly. This will result in greater computational effort to determine the optimal solution of this complete graph problem. Consider now the problem where a sparse graph is created with very low density such that no tour exists within the graph. If there are many assignment solutions with distinct costs in this problem, then much computational effort will be expended before all lower bounds approach infinity, and it is concluded that no tours exist. An example of this phenomenon is suggested by the entry in Table 3.9 for a density of .6. In the 20 replications of 6-city, symmetric, non-Euclidean problems with density .6, there were approximately as many assignment problems solved as were solved in 20 replications of complete graph, 6-city problems. This phenomenon is seen to occur in all three classes of problems and in fact, in the data of Table 3.14 it is seen that the number of assignment problems solved for a density of .6 was greater than for any other level including the complete problem and that this maximum computational effort was devoted not to determining the lowest cost tour, but rather to determining the existence of a tour.

Next, consider the proximity of the sparse graph optimal solution to that of the complete graph optimal solution. When the relative computational effort is also taken into account, the importance of the sparse graph in its own right becomes obvious. From Table 3.22 it is found that the optimal solution to the sparse problem of density .8 is also the optimal solution to the complete graph problem. Yet the computational effort required to solve the sparse graph represents a 96 percent reduction

of the effort in solving the complete graph problem. From Table 3.21 it is seen that the sparse problem of density .9 has an optimal cost which is only 4 percent above that of the complete graph problem. Yet, the computational effort required for the sparse graph problem represents a 38 percent reduction of the effort in solving the complete problem. Table 3.24 lists several other examples which demonstrate the power and applicability of the sparcity concept.

The investigation of the ramifications of sparcity next centers on the number of assignment problems which need to be solved to determine the optimum. In each table of the previous sections the statistic: mean CPU seconds per assignment problem is presented. Given a specific problem type and number of nodes the statistic is observed to vary only a small amount. Exceptions to this phenomenon are given in Tables 3.7 and 3.15 where the statistic is seen to vary by 50 and 100 percent at different density levels. A possible explanation for this case is the existence of very difficult assignment problems in the initial stages of the algorithm which yield lower bounds very close to the optimal solution and which are followed by several relatively simple assignment problems which lead to optimality. One conclusion results which is intuitive. As the number of nodes increases, within each problem class, the mean CPU time per assignment problem also increases. This is expected since increasing computational effort is required in solving assignment problems of increasing size.

The application of the Bellmore-Malone algorithm to the problem can be thought of as scanning a tree. Some branches terminate while others lead to still more branches. It is observed that the difference in

computational effort due to the introduction of sparcity can be attributed to a difference in the fullness and depth of the associated tree.

Concluding, it has been demonstrated by the experience reported in this chapter that sparcity can have a significant impact on the computational effort required to, optimally, solve the traveling salesman problem. The next chapter presents a heuristic algorithm which capitalizes upon the sparcity concept in order to reduce the computational effort required to solve the complete problem yet produce a tour close to the optimum.

CHAPTER IV

DEVELOPMENT OF HEURISTIC ALGORITHMS

The sparse graph problem is important in its own right, as demonstrated in the previous chapter. This chapter is, in part, founded upon the application of the sparcity concept in treating the complete graph problem. Specifically, a heuristic algorithm is developed which employs sparse graphs. Also, the use of the Bellmore-Malone algorithm as a vehicle to investigate the properties of sparse graphs has inspired a heuristic which modifies the Bellmore-Malone branching rule, yet preserves its branch and bound concept. We recall the major advantage of heuristics is in the generation of feasible and, hopefully, good solutions with relative savings in computational effort. That is, savings when compared with the computational effort necessary to solve the problem optimally. The chief disadvantage of the heuristic is the probable sub-optimality of the solutions generated by the heuristic.

This chapter is organized into three sections. The first two sections motivate and illustrate the above heuristics and present substantial computational experience. The final section details the computational experience and evaluates the performance of the heuristics.

A Sparse Graph Heuristic

The sparse graph heuristic is very general in nature in that it, like the Bellmore-Malone algorithm, can be applied to all three of the major classes of problems previously considered. Computational experience

will indicate its superior applicability to the symmetric-Euclidean class.

Basic Concept

The application of the sparse graph concept to the complete graph problem attempts to artificially create a sparse graph from its complete counterpart such that the former requires less computational effort to solve, yet produces a tour which departs relatively little from the optimal tour. The heuristic employs a parameter, P which, we will say, represents a "proximity of disengagement". This term is defined below. Following is a detailed description of the algorithm on an n-city problem.

This is a branch and bound algorithm, which solves successive assignment problems. Certain unfavorable links are removed as each successive assignment problem is generated, unless the number of subtours falls below or equals the parameter P. No further links are removed in this case, and the algorithm proceeds on that branch according to the Bellmore-Malone procedure.

The Algorithm

Step 1: Define a variable Z which represents the lowest cost tour yet found. Initially, set Z at some sufficiently large value, M.

Step 2: Solve the assignment problem resulting from the initial cost matrix.

Step 3: If a tour results, set Z equal to the cost of the tour and proceed to step 13. Otherwise, proceed to step 4.

Step 4: Determine the minimal cost assignment solution node and the associated assignment solution. If a tie exists, resolve the conflict at random.

Step 5: If this minimal cost node has value greater than or equal to Z go to step 13.

Step 6: If the number of subtours is less than or equal to P proceed to step 8. Otherwise, proceed to step 7.

Step 7: Define L as the number of subtours present in the current assignment solution. Consider a node N_i, for i = 1, 2, ..., n. Disregard from consideration the subtour S_q $1 \leq q \leq L$ which contains node i. Consider some subtour S_m with nodes N_1, N_2, ..., N_{t_m}. Determine the minimum distance arc from node N_i to nodes N_1, N_2, ..., N_{t_m}. Each arc weight not equal to the minimum is prohibited. The representative cost matrix values are set equal to a sufficiently large value, M. This process is repeated for all m, $1 \leq m \leq L$, $m \neq q$. Next, the entire process is repeated for all nodes i, $1 \leq i \leq n$.

Step 8: Determine the least cardinality subtour in this assignment solution. If a tie exists, it is randomly broken. Let this cardinality be k.

Step 9: Create k new subproblems. Let the k nodes in the subtour be N_1, N_2, ..., N_k. Subproblem i is formulated by changing the cost of the arc (N_i, N_r) to M in the cost matrix, for r = 1, 2, ..., i - 1, i + 1, ..., k and i = 1, 2, ..., k. The cost matrix is that of the optimal assignment solution considered and M is an arbitrarily large number.

Step 10: Assign to each of the k new sub-problems a cost equal to the optimal solution of the assignment problem from which they were generated.

Step 11: Determine the minimal cost node and associated assignment problem. If a tie exists, it is randomly broken. Solve the assignment problem.

Step 12: If a tour results, set Z equal to the minimum of the current value of Z and the cost of the tour. Otherwise, the value of Z remains unchanged. Return to step 4.

Step 13: The algorithm terminates and provides a solution with cost given by the current value of Z.

The proximity of disengagement parameter, P disengages or "jumps over" step 7 of the algorithm whenever the number of subtours present in the current assignment solution is less than or equal to P.

The heuristic algorithm is identical to the Bellmore-Malone algorithm except for step 7. Step 7 determines the minimum cost of travel between a node and all other nodes in a subtour. All arcs which have cost greater than the minimum are prohibited. This is repeated for all subtours except the subtour containing the given node. The entire procedure is then repeated for all nodes. The spirit of the Bellmore-Malone procedure is that when an assignment solution contains subtours, all must eventually be broken by links between subtours. The current heuristic algorithm expedites this process by prohibiting costly links between subtours. These links are not prime candidates for inclusion in the optimal tour, since they are of higher cost than the minimum cost connecting arcs. At each assignment solution this sparse graph generation technique is employed, except when the number of subtours present

is less than or equal to the proximity of disengagement parameter.

It is hypothesized that for a given problem class and size, the sparse graph generating technique described above is only efficient as long as the number of subtours is relatively large. This is verified in the computational experience. As the number of subtours becomes small, the sparse graph generator becomes too inefficient and coarse to eliminate the subtours. Instead, the sparse graph generator, represented in Step 7 of the heuristic is disengaged when the number of subtours falls below or equals P. The problem is solved by the Bellmore-Malone procedure from this point until termination of the algorithm is achieved.

Sample Problem

An example problem which demonstrates the sparse graph heuristic is considered next. The initial cost matrix is presented in Table 4.1. This assignment problem is solved, yielding the matrix given in Table 4.2. The optimal assignment solution has cost 14 and consists of three 2-city subtours, namely (1, 6, 1), (2, 4, 2), and (3, 5, 3). The sparse graph generation technique is next employed. Node 1 is first considered and the subtour (1, 6, 1) is disregarded since it contains node 1. Consider the costs associated with the arcs (1, 2) and (1, 4) in the optimal assignment matrix. These are 7 and 3 respectively, of which, the minimum is 3. This represents the minimum cost entry into subtour (2, 4, 2) from node 1. The arc (1, 2) is removed and the associated element in the cost matrix is set at M, a sufficiently large number. Next, the subtour (3, 5, 3) is considered; the arcs (1, 3) and (1, 5) both have cost 0 and neither is removed, since both costs equal the minimum. Node 2 is next considered and the subtour (2, 4, 2) is disregarded. Since arc (2, 1) has

TABLE 4.1. The Initial Cost Matrix for the Sparse Graph Heuristic Example Problem.

-	10	2	5	3	3
10	-	3	2	10	5
2	3	-	8	2	9
5	2	8	-	8	2
3	10	2	8	-	8
3	5	9	2	8	-

TABLE 4.2. The Optimal Assignment Matrix for the Initial Cost Matrix Problem.

-	7	0	3	0	0
7	-	1	0	7	2
0	1	-	7	0	7
3	0	7	-	6	0
0	7	0	6	-	5
0	2	7	0	5	-

cost 7 which is greater than 2, the cost of arc (2, 6), the former, is removed. Similarly, the arc (2, 5) is removed. Next, node 3 is considered, and so forth, until all nodes are considered and all arcs removed, with associated cost matrix elements being prohibited. This results in the matrix given by Table 4.3. The Bellmore-Malone branching rule is next applied, creating two sub-problems. The assignment solutions to these sub-problems are both tours of cost 15, hence this is the heuristic solution. Application of the Bellmore-Malone algorithm confirms the optimal solution to have cost 15.

Computational Experience

Tables 4.4 through 4.15 present the computational experience pertaining to this sparse graph generating heuristic algorithm for the three major classes of problems. As will be seen, the performance characteristics of the heuristic will be different for each class of problem.

A Modified Branching Heuristic

The modified branching rule heuristic is also very general in nature; no limitations are placed on the classes of problems to which it may be applied. Unlike the sparse graph heuristic, its efficiency is equally high in all problem classes.

Basic Concept

This heuristic was motivated primarily by the Bellmore-Malone algorithm and has its major difference in the subtour elimination phase. The subtour elimination method in this heuristic is motivated by both intuition and by the linear programming formulation of the traveling salesman problem. A detailed description of the algorithm on an n-city problem follows.

TABLE 4.3. The Matrix Resulting from the Application of the Sparse Graph Generating Technique.

-	-	0	3	0	0
-	-	1	0	-	2
0	1	-	-	0	-
-	0	-	-	6	0
0	-	0	6	-	-
0	-	-	0	5	-

Table 4.4. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm.

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
3	0.000	- .711
2	1.526	-18.208
1	6.188	-14.794

8-City, Non-Symmetric Problems.

10 replications.

SEED = 4327

TABLE 4.5. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm.

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
4	0.000	1.023
3	0.000	1.151
2	0.000	.512
1	2.802	-12.724

10-City, Non-Symmetric Problems.

10 replications.

SEED = 975

TABLE 4.6. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm.

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
6	0.000	- .500
5	0.000	- .593
4	0.000	- .036
3	0.000	- .114
2	2.131	34.452
1	2.131	18.668

15-City, Non-Symmetric Problems. 10 replications. SEED = 1895

TABLE 4.7. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm.

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
9	0.000	.243
8	0.000	- .367
7	0.000	.599
6	0.000	.270
5	0.000	.529
4	0.000	.200
3	0.000	- .005
2	1.026	-17.725
1	1.532	-28.429

20-City, Non-Symmetric Problems.

10 replications.

SEED = 20097

TABLE 4.8. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm.

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
3	0.000	18.623
2	.258	21.398
1	.778	24.440

8-City, Symmetric Problems.

10 replications.

SEED = 433

TABLE 4.9. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm.

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
4	.138	1.291
3	.317	-2.865
2	.317	-2.260
1	2.191	-25.585

10-City, Symmetric Problems.

10 replications.

SEED = 30301

TABLE 4.10. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm.

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
6	0.000	12.575
5	.676	12.429
4	2.214	8.582
3	3.066	34.211
2	4.116	35.643
1	5.373	34.902

15-City, Symmetric Problems.

5 replications.

SEED = 455

TABLE 4.11. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm.

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
9	0.000	5.939
8	0.000	6.757
7	0.000	6.854
6	0.000	16.676
5	0.000	16.204
4	0.000	14.701
3	0.000	26.091
2	0.000	36.517
1	0.000	40.849

20-City, Symmetric Problems.

1 replication.

SEED = 197

TABLE 4.12. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
3	1.762	62.053
2	4.734	75.300
1	5.019	75.700

8-City, Euclidean Problems.

5 replications.

TABLE 4.13. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
4	.414	62.882
3	1.241	72.103
2	1.653	76.029
1	4.787	72.608

10-City, Euclidean Problems

10 replications.

TABLE 4.14. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
6	.511	71.361
5	.517	71.467
4	1.614	75.287
3	2.465	76.669
2	2.465	75.146
1	2.684	73.941

15-City, Euclidean Problems

3 replications.

TABLE 4.15. Comparison of Sparse Graph Heuristic and Bellmore-Malone Algorithm

P	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
9	.098	62.436
8	.098	62.385
7	.098	62.195
6	.098	62.292
5	.098	62.583
4	.098	75.119
3	.153	75.261
2	3.443	75.167
1	3.443	75.374

20-City, Euclidean Problems

1 replication.

The Algorithm

Step 1: Define a variable Z to represent the lowest cost tour yet found. Initially, set Z at some sufficiently large value M.

Step 2: Solve the assignment problem resulting from the initial cost matrix.

Step 3: If a tour results, set Z equal to the cost of the tour and proceed to Step 11. Otherwise, proceed to Step 4.

Step 4: Determine the minimal cost assignment solution node and the associated assignment solution. If a tie exists, it is randomly broken.

Step 5: If this minimal cost node has value greater than or equal to Z go to step 11.

Step 6: Determine the least cardinality subtour in this assignment solution. If a tie exists, it is randomly broken. Let this cardinality be k.

Step 7: Create k new assignment problems. Let the k nodes in the subtour be N_1, N_2, ..., N_k, N_{k+1}, where $N_{k+1} = N_1$. Subproblem i ($1 \leq i \leq k$) is formulated by changing the cost of the arc (N_i, N_{i+1}) to M and fixing the k-1 other arcs in the optimal solution. This is achieved by changing all entries in rows N_1, N_2, ..., N_k, except row N_i, to M, and all columns N_1, N_2, ..., N_k, except column N_{i+1}, to M. Next the cost matrix entries representing the k-1 arcs which are fixed are reset to zero. The cost matrix is that of the optimal assignment solution considered and M is an arbitrarily large number.

Step 8: Assign to each of the k new subproblems a cost equal to the optimal solution of the assignment problem from which they were generated.

Step 9: Determine the minimal cost node and associated assignment problem. If a tie exists, it is randomly broken. Solve the assignment problem.

Step 10: If a tour results, set Z equal to the minimum of the current value of Z and the cost of the tour. Otherwise, the value of Z remains unchanged. Return to Step 4.

Step 11: The algorithm terminates and provides a solution with cost given by the current value of Z.

The major difference between this heuristic and the Bellmore-Malone algorithm is in the generation of sub-problems. The latter prohibits certain arcs from consideration at each assignment solution, whereas the heuristic fixes certain arcs in the final tour. The dimension of the subproblems are less than that of the original problem with the heuristic and also, the dimension of successive generation subproblems decreases.

The motivation of the heuristic is in the linear programming formulation of the problem given in Chapter II. The dual of this problem is given below.

$$\text{Maximize} \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j + \sum_{L=1}^{M} (|C_L| - 1)w_L$$

subject to $u_i + v_j + w_L \leq c_{ij}$ for i, j = 1, 2, ..., n and w_L such that subtour L contains arc (i, j). L = 1, 2, ..., M where M is the total number of

subtours possible.

u_i, v_j unrestricted.

$w_i \leq 0$.

The dual variables u_i, v_j correspond to the constraints given by equations (1) and (2) in Chapter II respectively. The dual variables w_L correspond to the subtour constraints (3) in Chapter II.

Complementary slackness conditions specify that

$$(|C_L| - 1 - \sum_{(i,j)\in C_L} x_{ij}^*)w_L^* = 0 \qquad (1)$$

for each possible subtour L, L = 1, 2, ..., M. Values x_{ij}^* and w_L^* are the optimal values of the primal and dual variables respectively.

Consider the solution of the initial assignment problem. If the assignment solution contains no subtours then this solution represents a primal feasible solution and is therefore the optimal solution to the traveling salesman problem. However, if no tour is present then the assignment solution violates at least one subtour constraint. When interpreted in a linear programming sense, the dual variable corresponding to the subtour constraint will next enter the basis. Of course, this dual variable may or may not be in the final optimal basis, however the heuristic algorithm fixes this variable in the basis. Equation (1) now specifies that

$$|C_L| - 1 - \sum_{(i,j)\in C_L} x_{ij}^* = 0$$

or equivalently,

$$\sum_{(i,j)\in C_L} x^*_{ij} = |C_L| - 1 \qquad (2)$$

This provides the basis for generating k new subproblems each with a distinct set of k-1 arcs being fixed.

Sample Problem

The same example problem used to demonstrate the sparse graph heuristic will be used to demonstrate the modified branching rule heuristic. The initial cost matrix and initial assignment matrix are given in Tables 4.1 and 4.2 respectively. The modified branching heuristic determines the least cardinality subtour to have cardinality 2. The initial assignment solution is (1, 6, 1), (2, 4, 2), and (3, 5, 3). The subtour (1, 6, 1) is arbitrarily chosen and two subproblems are created from the optimal assignment cost matrix. One subproblem has the arc (1, 6) fixed in the solution with arcs (6, 1), (1, Y), and (Y, 6) $1 \leq Y \leq 6$ removed and the associated cost elements prohibited. The other subproblem fixes arc (6, 1) and removes arcs (1, 6), (6, Y), and (Y, 1) $1 \leq Y \leq 6$. The associated cost matrices are given in Tables 4.16 and 4.17. Both subproblems have optimal assignment solutions which are tours of cost 15, hence, the heuristic solution is again optimal in this example.

Computational Experience

Another motivation for the heuristic is that whenever an arc is prohibited, it must be replaced by another arc which may or may not have cost zero in the assignment matrix. It is desirable to adopt a rule which preserves as many arcs of zero cost as possible in the final tour and the heuristic algorithm effects this notion. The computational experience

TABLE 4.16. The Modified Branching Rule Demonstrated

-	-	-	-	-	0
7	-	1	0	7	-
0	1	-	7	0	-
3	0	7	-	6	-
0	7	0	6	-	-
-	2	7	0	5	-

TABLE 4.17. The Modified Branching Rule Demonstrated

-	7	0	3	0	-
-	-	1	0	7	2
-	1	-	7	0	7
-	0	7	-	6	0
-	7	0	6	-	5
0	-	-	-	-	-

as applied to this heuristic for the three major classes of problems is given in Tables 4.18 through 4.20.

Discussion of Computational Results

Analysis of the computational results of the sparse graph heuristic reveals that its application produces tours which are very close to optimal for the classes and sizes of problems considered. This holds for all values of the proximity disengagement parameter. The worst performance is approximately 6 percent above the optimal. However, the percent reduction in computational effort is significant only in symmetric-Euclidean problems, and symmetric, non-Euclidean problems with 15 or 20 nodes (cities). The savings in computational effort is especially large in the symmetric-Euclidean problems where reduction ranged from 60 to 80 percent for all problem sizes considered. In the 20-city symmetric, non-Euclidean problem referenced in Table 4.11, the sparse graph heuristic achieves the optimal solution with a percent reduction in computational effort which exceeds 40 percent.

In the non-symmetric problems as well as for smaller symmetric, non-Euclidean ones, it is found that the heuristic may produce sub-optimal tours with computational effort greater than that of the optimal procedure. The explanation for this phenomenon can be attributed to the fact that the heuristic procedure is based on Euclidean notions. That is, larger distance arcs are prohibited. The non-symmetric problems do not lend themselves to this Euclidean abstraction and the smaller symmetric problems require much computational effort to create the sparse graph, yet in its creation, few arcs are prohibited due to the large number of subtours created.

TABLE 4.18. Comparison of Modified Branching Rule Heruistic and Bellmore-Malone Algorithm.

Number of Cities	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
6	0.000	24.561
7	1.070	30.238
8	.985	27.312
9	0.000	27.086
10	.405	46.228
15	0.000	47.737
20	0.000	3.423

Non-Symmetric Problems

10 replications.

TABLE 4.19. Comparison of Modified Branching Rule Heuristic and Bellmore-Malone Algorithm

Number of Cities	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
6	1.013	25.510
7	0.000	25.216
8	1.786	39.968
9	0.000	41.059
10	.475	21.486
15	.613	31.327
20	.781	54.167

Symmetric Problems.

10 replications.

TABLE 4.20. Comparison of Modified Branching Rule Heuristic and Bellmore-Malone Algorithm

Number of Cities	Percent Above Mean Optimal	Percent Reduction in CPU Seconds
6	0.000	15.188
7	0.000	22.601
8	1.545	40.705
9	0.000	56.046
10	0.000	54.222
15	0.000	77.245
20	0.000	46.992

Euclidean Problems.

5 replications.

It was earlier hypothesized that the sparse graph generation technique was not efficient when the number of subtours became small. The computational experience verifies this assertion as can be seen from the data where P is equal to 1 or 2. For non-symmetric and symmetric, non-Euclidean problems the computational effort is adversely affected and for symmetric-Euclidean problems the solution quality degenerates.

The modified branching rule heuristic surpasses the sparse graph heuristic in both percent reduction in computational effort and percent above the optimal solution. The modified branching rule heuristic achieves the optimal solution in a majority of the problems treated. The percent above the mean optimal solution is under 2 percent for all classes and problem sizes treated, along with percent reductions in computational effort in the range of 20 to 50 percent. All problems in which the percent above the mean optimal exceeded 1 percent, resulted in a percent reduction in computational effort of no less than 25 percent. From Tables 4.18 and 4.20 it can be seen that for problem sizes of 15 and 20 nodes the optimal solution was achieved in no fewer than 5 replications with percent reductions in computational effort of approximately 47 and 77 percent. The heuristic appears to apply equally well to all three classes of problems and unlike the sparse graph heuristic the computational effort required is always less than that of the optimal procedure. The reduction in computational effort is dramatic such that in only 2 cases was the percent reduction less than 20 percent and the optimal mean cost was achieved in both cases.

CHAPTER V

CONCLUSIONS AND EXTENSIONS

The fundamental objective of this thesis has been to examine the impact of graph sparcity on the solution of traveling salesman problems. Specifically, many problems of various degrees of sparcity were constructed and solved using a well-known, exact algorithm developed by Bellmore and Malone. It has been demonstrated that the application of the Bellmore-Malone algorithm to the sparse graph problem results in computational effort which varies greatly. This suggests that possible modifications to the algorithm in order to adapt it to the sparse graph problem might reduce the computational effort involved in sparse graph problems.

One major achievement of this thesis is the demonstration of sparcity as a non-trivial concept. Computational experience with sparse graph problems establishes the sparse problem as being important in its own right. Of nearly equal importance, it has been demonstrated that the sparse graph problem may be viewed as a vehicle with which to treat the complete graph problem itself. One method has been proposed to create sparse graphs from the initial complete graph problem. This was manifested in a sparse graph heuristic algorithm. The performance of this algorithm was found to be excellent in the case of symmetric-Euclidean problems and marginal for the other classes of problems. Yet this algorithm was motivated by Euclidean considerations, hence its success in the area of Euclidean problems tends to overshadow the mediocrity evidenced in the smaller problems of the other classes. The power

of sparse graph generation in treating the complete problem is evidenced by the sparse graph heuristic. It is speculated that, as a further extension of the sparse graph concept, algorithms may be developed which create sparse graphs from complete ones for the other two classes of problems, such that a substantial reduction in computational effort results with only a small departure from optimality.

A modified branching rule heuristic was developed after being motivated by linear programming considerations. It has similarities as well as differences with respect to both Little's procedure[24] and the Bellmore-Malone algorithm. Like the Bellmore-Malone algorithm, the heuristic solves a sequence of assignment problems. It is not a subtour elimination method however, but rather a tour building algorithm, as is Little's work. Little's procedure solves no assignment problems, however an optimal tour may be formed at the expense of much computational effort. It is believed that the solution of assignment problems is a very efficient sub-process of the algorithm and this combined with the power of the tour building approach, in order to reduce the dimension of the problem, results in the overall efficiency of the modified branching rule heuristic.

The solution of successive assignment problems is a fundamental step in all of the algorithms considered in this thesis. As such, the role of the assignment solution algorithm cannot be minimized. The solution technique employed in this research is the Hungarian algorithm, which is probably the traditional mode of solution. The effects of sparcity, and the performance of the heuristics presented here when another assignment solution algorithm is employed is clearly a subject

of further investigation. For example, another assignment algorithm which could be implemented in the work of Bellmore and Malone is the modified transportation algorithm of Ford and Fulkerson.[14]

The limitation of computer resources prevented the extension of this work to larger problems on a general level. It was possible to extend the computational experience with the heuristics to a level of 40-city non-symmetric problems; however, the computational effort necessary to optimally solve the problem was prohibitive. Consequently, no evaluation was possible. Generally, most heuristic procedures decrease in efficiency as problem size increases. This is not the case for problems treated with the sparse graph heuristic, in which marginal efficiency was achieved for small non-symmetric and symmetric, non-Euclidean problems; yet the efficiency increased as the largest problem sizes were considered in this class.

Recapping the major points of this thesis, the traveling salesman problem is one of the most important problems encountered in discrete optimization, with widespread applications in other areas such as job shop scheduling. Any heuristic solution technique will be required to produce tours which are sufficiently near the optimum in cost, yet the computational effort must be minimal. The exact algorithms produce the optimal solution but at the expense of greatly increased computational effort.

The results of the research into sparse graph problems demonstrate that they exhibit very interesting properties and may be employed as a tool for solving the complete graph problem. Specifically, through the artificial creation of sparse graphs, near optimal tours may be produced

with computational effort significantly less than that of current exact techniques. Clearly, additional work in this area alone is warranted.

APPENDIX

A source program listing is contained in the appendix.

1. Sparse Graph Solution Program.
2. Sparse Graph Heuristic Program.
3. Modified Branching Heuristic Program.

```
C       THIS PROGRAM CREATES AND SOLVES SPARSE GRAPH
C        PROBLEMS.  IT REPLICATES THESE PROBLEMS FOR
C        A GIVEN SIZE AND PROBLEM CLASS, AND DETERMINES
C        THE MEAN OPTIMAL SOLUTION, MEAN COMPUTATIONAL
C        EFFORT, MEAN NUMBER OF ASSIGNMENT PROBLEMS
C        SOLVED, AND CALCULATES MEAN COMPUTATIONAL
C        EFFORT PER ASSIGNMENT PROBLEM SOLVED.
      PROGRAM MAIN(YES,INPUT,OUTPUT,PARK30,GARBAG,EXP,FIN,
     *  ZIPP,
     *   UU,SMAK,
     *   TAPE22=UU,TAPE2=SMAK,
     *  TAPE5=YES,TAPE6=EXP,TAPE8=GARBAG,TAPE7=PARK30,
     *  TAPE21=ZIPP,
     *  TAPE10=INPUT,TAPE11=OUTPUT,TAPE16=FIN)
      COMMON /LAB1/MAT(50,50),IDEL(50,50),IR(50),IC(50),
     *   IX1(50),IX2(50),ISLIP(300)
      COMMON /LAB5/MAT2(50,50),IBM(50),XMAT7(9,4)
      COMMON /LAB6/MUCK(300),MUCK2(300)
      COMMON /LAB7/ISEED,SPR,NUMCIT,IUP
      COMMON /LAB8/IUEW
      REWIND 8
      PRINT*," HOW MANY CITIES --- AND WHAT SEED"
      PRINT*," ---AND WHAT UPPER LIMIT"
      PRINT*," ---AND HOW MANY REPLICATIONS"
      READ(10,*)NUMCIT,ISEED,IUP,IREPS
      IDY=ISEED
      DO 7777 IUEW=1,IREPS
      ISEED=IDY+22*(IUEW-1)
      CALL SPUT2
      DO 3339 IGFT=1,9
      SPR=.1+.1*IGFT
      CALL ASS23
      YYUKK=SECOND(1.)
      ITINT=0
      MMIN=99999
98849 CONTINUE
      INTC=0
      REWIND 5
      REWIND 21
      REWIND 6
  650 READ(5,*)IDSUM
      IF(EOF(5))750,751
  751 CONTINUE
      READ(5,*)((MAT(I,J),J=1,NUMCIT),I=1,NUMCIT)
      DO 64 I=1,NUMCIT
      MIN=99999
      DO 66 J=1,NUMCIT
      IF(MAT(I,J).LT.MIN  .AND. MAT(I,J).GE.0)MIN=MAT(I,J)
   66 CONTINUE
      IF(MIN.EQ.0) GO TO 64
```

```
      DO 68 J=1,NUMCIT
      MAT(I,J)=MAT(I,J)-MIN
   68 CONTINUE
      IR(I)=IR(I)+MIN
      IDSUM=IDSUM+MIN
   64 CONTINUE
      DO 70 J=1,NUMCIT
      MIN=99999
      DO 72 I=1,NUMCIT
      IF(MAT(I,J).LT.MIN  .AND.  MAT(I,J).GE.0)MIN=MAT(I,J)
   72 CONTINUE
      IF(MIN.EQ.0) GO TO 70
      DO 74 I=1,NUMCIT
      MAT(I,J)=MAT(I,J)-MIN
   74 CONTINUE
      IC(J)=IC(J)+MIN
      IDSUM=IDSUM+MIN
   70 CONTINUE
   75      ICOUNT=0
           DO 81 K=1,NUMCIT
           IX1(K)=0
           IX2(K)=0
   81 CONTINUE
           DO 187 K2=1,NUMCIT
           DO 188 K3=1,NUMCIT
           IDEL(K2,K3)=0
  188 CONTINUE
  187 CONTINUE
   13 FORMAT(1X,9I8/9(4X,9I8/))
   77 NIDUM=0
      DO 78 J=1,NUMCIT
      IDUM=0
      DO 80 I=1,NUMCIT
      IF(MAT(I,J).NE.0) GO TO 80
      IDUM=IDUM+1
      ILOC=I
   80 CONTINUE
      IF(IDUM.EQ.1) GO TO 82
      IF(IDUM.EQ.0)NIDUM=NIDUM+1
   78 CONTINUE
      DO 84 I=1,NUMCIT
      IDUM=0
      DO 86 J=1,NUMCIT
      IF(MAT(I,J).NE.0) GO TO 86
      IDUM=IDUM+1
      ILOC=J
   86 CONTINUE
      IF(IDUM.EQ.1) GO TO 88
      IF(IDUM.EQ.0)NIDUM=NIDUM+1
   84 CONTINUE
```

```
      IF(NIDUM.EQ.2*NUMCIT) GO TO 104
      DO 332 M2=1,NUMCIT
      IDUM=0
      DO 334 M4=1,NUMCIT
      IF(MAT(M2,M4).NE.0) GO TO 334
      IDUM=IDUM+1
      MUCK(IDUM)=M4
  334 CONTINUE
      IF(IDUM.EQ.2) GO TO 336
      GO TO 332
  336 IDUM=0
  337 IDUM=IDUM+1
      IDUM2=0
      DO 340 M8=1,NUMCIT
      IF(MAT(M8,MUCK(IDUM)).NE.0) GO TO 340
      IDUM2=IDUM2+1
      MUCK2(IDUM2)=M8
  340 CONTINUE
      IF(IDUM2.NE.2  .AND.  IDUM.EQ.1) GO TO 337
      IF(IDUM.EQ.2) GO TO 332
      GO TO 342
  332 CONTINUE
      PRINT*," NO TWO ZEROES IN ROW AND COL"
      PRINT*," GO TO 650"
      GO TO 650
  342 ILOC=M2
      J=MUCK(IDUM)
  567 DO 344 M10=1,NUMCIT
      IF(MAT(M10,J).EQ.0) MAT(M10,J)=-88888
  344 CONTINUE
      GO TO 82
  104 CONTINUE
      DO 108 I=1,NUMCIT
      DO 110 J=1,NUMCIT
      IF(MAT(I,J).EQ.-88888)MAT(I,J)=0
  110 CONTINUE
  108 CONTINUE
      ISUM=0
      DO 112 I=1,NUMCIT
      DO 114 J=1,NUMCIT
      ISUM=ISUM+IDEL(I,J)
  114 CONTINUE
  112 CONTINUE
      IF(ISUM.EQ.NUMCIT**2) GO TO 116
      IMIN=99999
      DO 118 I=1,NUMCIT
      DO 120 J=1,NUMCIT
      IF( IDEL(I,J).NE.0) GO TO 120
      IF(MAT(I,J).LT.IMIN  .AND.  MAT(I,J).GT.0)IMIN=MAT(I,
     *  J)
```

```
 120 CONTINUE
 118 CONTINUE
     DO 122 I=1,NUMCIT
     DO 124 J=1,NUMCIT
     IF(IDEL(I,J).EQ.0) MAT(I,J)=MAT(I,J)-IMIN
     IF(IDEL(I,J).EQ.2)MAT(I,J)=MAT(I,J)+IMIN
 124 CONTINUE
 122 CONTINUE
     DO 233 IL=1,NUMCIT
     DO 234 IM=1,NUMCIT
     IF(MAT(IL,IM).GE.0)GO TO 234
     PRINT*," NEGATIVE  NUMBER  IN  MATRIX------>  ABORT"
     PRINT*," GO TO 650"
     GO TO 650
 234 CONTINUE
 233 CONTINUE
     IDSUM=IDSUM+(NUMCIT-ICOUNT)*IMIN
     GO TO 75
  82 ICOUNT=ICOUNT+1
     IX1(ICOUNT)=ILOC
     IX2(ICOUNT)=J
     DO 128 I2=1,NUMCIT
     IDEL(ILOC,I2)=IDEL(ILOC,I2)+1
     IF(MAT(ILOC,I2).EQ.0)MAT(ILOC,I2)=-88888
 128 CONTINUE
     MAT(ILOC,J)=-88888
     GO TO 77
  88 ICOUNT=ICOUNT+1
     IX1(ICOUNT)=I
     IX2(ICOUNT)=ILOC
     DO 130 I2=1,NUMCIT
     IDEL(I2,ILOC)=IDEL(I2,ILOC)+1
     IF(MAT(I2,ILOC).EQ.0)MAT(I2,ILOC)=-88888
 130 CONTINUE
     MAT(I,ILOC)=-88888
     GO TO 77
 116 CONTINUE
     DO 2222 IV=1,NUMCIT
     IF(IX1(IV).NE.IX2(IV)) GO TO 2222
     PRINT*," *****",IX1(IV),IX2(IV)
     GO TO 650
2222 CONTINUE
     IF(IDSUM.GE.MMIN) GO TO 650
     IHA=3*NUMCIT
     DO 701 IL=1,IHA
     ISLIP(IL)=0
 701 CONTINUE
     ISLIP(1)=1
     ISTOP=1
     IF=1
```

```
      IML=99999
720   ICT=1
702   ILOOK=ISLIP(IF)
703   DO 704 IL=1,NUMCIT
      IF2=IL
      IF(IX1(IL).EQ.ILOOK) GO TO 705
704   CONTINUE
705   ISLIP(IF+1)=IX2(IF2)
      ICT=ICT+1
      IF=IF+1
      IF(ISLIP(IF).EQ.ISTOP) GO TO 706
      GO TO 702
706   IF=IF+1
      IF(ICT.EQ.NUMCIT+1) GO TO 790
      IF(ICT.LT.IML) IML=ICT
      ISLIP(IF)=-44444
      DO 707 IL=1,NUMCIT
      DO 708 IM=1,IF
      IF(ISLIP(IM).EQ.IL) GO TO 707
708   CONTINUE
      ISLIP(IF+1)=IL
      IF=IF+1
      ISTOP=IL
      GO TO 720
707   CONTINUE
      IK=0
      IP=0
709   IK=IK+1
      IF(ISLIP(IK).EQ.-44444) GO TO 710
      IP=IP+1
      IBM(IP)=ISLIP(IK)
      GO TO 709
710   IF(IP.EQ.IML) GO TO 711
      IP=0
      GO TO 709
711   DO 712 IL=1,NUMCIT
      DO 713 IM=1,NUMCIT
      MAT2(IL,IM)=MAT(IL,IM)
713   CONTINUE
712   CONTINUE
      IMM=IML-1
      DO 717 IL=1,IMM
          DO 715 IL2=1,NUMCIT
          DO 716 IM2=1,NUMCIT
          MAT(IL2,IM2)=MAT2(IL2,IM2)
716   CONTINUE
715   CONTINUE
      DO 718 IM=1,IMM
      IF(IL.EQ.IM) GO TO 718
      MAT(IBM(IL),IBM(IM))=99999
```

```
  718 CONTINUE
      INTC=INTC+1
      WRITE(21,13)IDSUM
      DO 719 IM=1,NUMCIT
      WRITE(21,13)(MAT(IM,IN),IN=1,NUMCIT)
  719 CONTINUE
  717 CONTINUE
       GO TO 650
  750 CALL RETURN("YES")
      ITINT=ITINT+INTC
      ENDFILE 21
      REWIND 21
      CALL RENAME("YES","ZIPP")
      REWIND 5
      READ(5,*)IDUM
      IF(EOF(5))791,792
  792 GO TO 98849
  790 CONTINUE
      IF(IDSUM.LT.MMIN) MMIN=IDSUM
      GO TO 650
  791 CONTINUE
      HTM=SECOND(1.)-YYUKK
      XMAT7(IGFT,1)=MMIN
      XMAT7(IGFT,2)=HTM
      XMAT7(IGFT,3)=ITINT+1
      XMAT7(IGFT,4)=XMAT7(IGFT,2)/XMAT7(IGFT,3)
 3339 CONTINUE
      PRINT*,"    "
      PRINT*," AT NUMBER-----",IUEW
      WRITE(11,16)((XMAT7(IL,IM),IM=1,4),IL=1,9)
      WRITE(8,16)((XMAT7(IL,IM),IM=1,4),IL=1,9)
   16 FORMAT(//9(1X,F15.0,F15.3,F15.0,F15.3/))
 7777 CONTINUE
      STOP
      CALL STAT
      STOP
      END



      SUBROUTINE SPUT2
      DIMENSION MAT(50,50)
      COMMON /LAB7/ISEED,SPR,NUMCIT,IUP
      REWIND 2
      READ(2,*)((MAT(IL,IM),IM=1,NUMCIT),IL=1,NUMCIT)
      DO 36 I=1,NUMCIT
      MAT(I,I)=999999
```

```
 36 CONTINUE
    REWIND 22
    WRITE(22,14) 0
 14 FORMAT(1X,I5)
    DO 42 I=1,NUMCIT
    WRITE(22,13)(MAT(I,J),J=1,NUMCIT)
 13 FORMAT(1X,10I7/9(4X,9I7/))
 42 CONTINUE
    ENDFILE 22
    REWIND 22
    RETURN
    END




    SUBROUTINE ASS23
    DIMENSION MAT(50,50)
    COMMON /LA87/ISEED,SPR,NUMCIT,IUP
    REWIND 22
    READ(22,*)IDUM
    READ(22,*)((MAT(IL,IM),IM=1,NUMCIT),IL=1,NUMCIT)
    CALL RANSET(ISEED)
    IOH=.5*((1-SPR)*NUMCIT*(NUMCIT-1))+1
    IF(SPR.EQ.1.) GO TO 931
    IPPP=0
    IL88=50*NUMCIT*(NUMCIT-1)
    DO 31 IL=1,IOH
 30 I1=NUMCIT*RANF(1.0)+1
    I2=NUMCIT*RANF(1.0)+1
    IPPP=IPPP+1
    IF(IPPP.LT.IL88) GO TO 1094
    PRINT*," DESIRED SPARCITY IMPOSSIBLE"
    GO TO 931
1094 CONTINUE
    IF(I1.EQ.I2) GO TO 30
    IA1=0
    IA2=0
    DO 40 IL1=1,NUMCIT
    IF(MAT(I1,IL1).EQ.99999.OR.MAT(I1,IL1).EQ.999999
   *  ) GO TO 41
    IA1=IA1+1
 41 IF(MAT(IL1,I2).EQ.99999 .OR. MAT(IL1,I2).EQ.
   *   999999) GO TO 40
    IA2=IA2+1
 40 CONTINUE
    IF(IA1.EQ.1  .OR.  IA2.EQ.1) GO TO 30
    IF(MAT(I1,I2).EQ.99999) GO TO 30
```

```
      MAT(I1,I2)=99999
   31 CONTINUE
  931 REWIND 5
      WRITE(5,13)0
      DO 32 IL=1,NUMCIT
      WRITE(5,13)(MAT(IL,IM),IM=1,NUMCIT)
      WRITE(11,13)(MAT(IL,IM),IM=1,NUMCIT)
   13 FORMAT(1X,10I7/9(4X,9I7/))
   32 CONTINUE
      PRINT*,"     "
      ENDFILE 5
      REWIND 5
      RETURN
      END




      SUBROUTINE STAT
      DIMENSION XMAT2(9,4),XMAT3(9,4),XMAT4(9,4),XMAT(9,4)
      DATA XMAT2/36*0./,XMAT3/36*0./
      PRINT*,"################################"
      PRINT*,"  "
      IDT=0
      REWIND 8
   34 READ(8,*)((XMAT(IM,IN),IN=1,4),IM=1,9)
      IF(EOF(8))31,32
   32 CONTINUE
      IDT=IDT+1
      DO 33 IM=1,9
      DO 33 IN=1,4
      XMAT2(IM,IN)=XMAT2(IM,IN)+XMAT(IM,IN)
      XMAT3(IM,IN)=XMAT3(IM,IN)+XMAT(IM,IN)**2
   33 CONTINUE
      GO TO 34
   31 DO 35 IM=1,9
      DO 35 IN=1,4
      XMAT2(IM,IN)=XMAT2(IM,IN)/IDT
      XMAT3(IM,IN)=((XMAT3(IM,IN)-IDT*XMAT2(IM,IN)**2)/(
     *    IDT-1))**.5
   35 CONTINUE
      DO 36 IM=1,9
      DO 36 IN=1,4
      XMAT4(IM,IN)=XMAT2(IM,IN)-1.96*XMAT3(IM,IN)/IDT**.5
   36 CONTINUE
      DO 37 IM=1,9
      WRITE(11,13)(XMAT4(IM,IN),IN=1,4)
   13 FORMAT(1X,4F11.3)
```

```
37 CONTINUE
   PRINT*,"  "
   DO 38 IM=1,9
   WRITE(11,13)(XMAT2(IM,IN),IN=1,4)
38 CONTINUE
   DO 39 IM=1,9
   DO 39 IN=1,4
   XMAT4(IM,IN)=XMAT2(IM,IN)+1.96*XMAT3(IM,IN)/IDT**.5
39 CONTINUE
   PRINT*,"  "
   PRINT*," WHEN ALL NUMBERS ARE COPIED ENTER 1"
   READ(10,*)IIIC
   DO 40 IM=1,9
   WRITE(11,13)(XMAT4(IM,IN),IN=1,4)
40 CONTINUE
   RETURN
   END
```

```
C    THIS IS THE HEURISTIC WHICH DOES A BELLMORE-MALONE
C     AND ALSO PROHIBITS ALL NODES TO OTHER NODES
C     EXCEPT THE NODE IN EACH SUBTOUR CLOSEST TO THE GIVEN
C     NODE
      PROGRAM MAIN(YES,INPUT,OUTPUT,MF,GARBAG,EXP,FIN,ZIPP,
     *    UU,PARK30,
     *    TAPE22=UU,TAPE7=PARK30,
     *   TAPE5=YES,TAPE6=EXP,TAPE8=GARBAG,TAPE9=MF,TAPE21=
     *   ZIPP,
     *   TAPE10=INPUT,TAPE11=OUTPUT,TAPE16=FIN)
      COMMON /LAB1/MAT(50,50),ICEL(50,50),IR(50),IC(50),
     *    IX1(50),IX2(50),ISLIP(300),IBM2(57)
      COMMON /LAB5/MAT2(50,50),IBM(50)
      COMMON /LAB6/MUCK(300),MUCK2(300)
      COMMON /LAB7/ISEED,SPR,NUMCIT,IUP
      COMMON /LAB8/IUEW
      COMMON /LAB9/MURF(50,2)
      REAL MURF
      DATA MURF/100*0./
      PRINT*," HOW MANY CITIES???"
      PRINT*," WHAT IS SEED ..... UPPER LIMIT"
      PRINT*," PROXIMITY DISENGAGEMENT?"
      PRINT*," NUMBER OF REPS"
      READ(10,*)NUMCIT,ISEED,IUP,IKART,ITER
      DO 7114 IQQQ=1,ITER
      PRINT*," AT # ",IQQQ
      ISEED=ISEED+3
      IUEW=ISEED
      INART=IKART-1
      IZO=0
      DO 7113 IPPS=IZO,INART
      YYUKK=SECOND(1.)
      ISKNK=1
      MMIN=99999
      IYART=IKART-IPPS
      CALL SPUT2
98849 CONTINUE
      INTC=0
      REWIND 8
      REWIND 9
      REWIND 5
      REWIND 21
      REWIND 6
  650 READ(5,*)IDSUM
      IF(EOF(5))750,751
  751 CONTINUE
      READ(5,*)((MAT(I,J),J=1,NUMCIT),I=1,NUMCIT)
      DO 64 I=1,NUMCIT
      MIN=99999
      DO 66 J=1,NUMCIT
```

```
      IF(MAT(I,J).LT.MIN  .AND. MAT(I,J).GE.0)MIN=MAT(I,J)
   66 CONTINUE
      IF(MIN.EQ.0) GO TO 64
      DO 68 J=1,NUMCIT
      MAT(I,J)=MAT(I,J)-MIN
   68 CONTINUE
      IR(I)=IR(I)+MIN
      IDSUM=IDSUM+MIN
   64 CONTINUE
      DO 70 J=1,NUMCIT
      MIN=99999
      DO 72 I=1,NUMCIT
      IF(MAT(I,J).LT.MIN  .AND.  MAT(I,J).GE.0)MIN=MAT(I,J)
   72 CONTINUE
      IF(MIN.EQ.0) GO TO 70
      DO 74 I=1,NUMCIT
      MAT(I,J)=MAT(I,J)-MIN
   74 CONTINUE
      IC(J)=IC(J)+MIN
      IDSUM=IDSUM+MIN
   70 CONTINUE
      IF(IDSUM.GE.MMIN) GO TO 650
   75      ICOUNT=0
           DO 81 K=1,NUMCIT
           IX1(K)=0
           IX2(K)=0
   81 CONTINUE
           DO 187 K2=1,NUMCIT
           DO 188 K3=1,NUMCIT
           IDEL(K2,K3)=0
  188 CONTINUE
  187 CONTINUE
   13 FORMAT(1X,9I8/9(4X,9I8/))
   77 NIDUM=0
      DO 78 J=1,NUMCIT
      IDUM=0
      DO 80 I=1,NUMCIT
      IF(MAT(I,J).NE.0) GO TO 80
      IDUM=IDUM+1
      ILOC=I
   80 CONTINUE
      IF(IDUM.EQ.1) GO TO 82
      IF(IDUM.EQ.0)NIDUM=NIDUM+1
   78 CONTINUE
      DO 84 I=1,NUMCIT
      IDUM=0
      DO 86 J=1,NUMCIT
      IF(MAT(I,J).NE.0) GO TO 86
      IDUM=IDUM+1
      ILOC=J
```

```
 86 CONTINUE
    IF(IDUM.EQ.1) GO TO 88
    IF(IDUM.EQ.0)NIDUM=NIDUM+1
 84 CONTINUE
    IF(NIDUM.EQ.2*NUMCIT) GO TO 104
    DO 332 M2=1,NUMCIT
    IDUM=0
    DO 334 M4=1,NUMCIT
    IF(MAT(M2,M4).NE.0) GO TO 334
    IDUM=IDUM+1
    MUCK(IDUM)=M4
334 CONTINUE
    IF(IDUM.EQ.2) GO TO 336
    GO TO 332
336 IDUM=0
337 IDUM=IDUM+1
    IDUM2=0
    DO 340 M8=1,NUMCIT
    IF(MAT(M8,MUCK(IDUM)).NE.0) GO TO 340
    IDUM2=IDUM2+1
    MUCK2(IDUM2)=M8
340 CONTINUE
    IF(IDUM2.NE.2  .AND.  IDUM.EQ.1) GO TO 337
    IF(IDUM.EQ.2) GO TO 332
    GO TO 342
332 CONTINUE
    PRINT*," NO TWO ZEROES IN ROW AND COL"
    PRINT*," GO TO 650"
    GO TO 650
342 ILOC=M2
    J=MUCK(IDUM)
567 DO 344 M10=1,NUMCIT
    IF(MAT(M10,J).EQ.0) MAT(M10,J)=-88888
344 CONTINUE
    GO TO 82
104 CONTINUE
    DO 108 I=1,NUMCIT
    DO 110 J=1,NUMCIT
    IF(MAT(I,J).EQ.-88888)MAT(I,J)=0
110 CONTINUE
108 CONTINUE
    ISUM=0
    DO 112 I=1,NUMCIT
    DO 114 J=1,NUMCIT
    ISUM=ISUM+ICEL(I,J)
114 CONTINUE
112 CONTINUE
    IF(ISUM.EQ.NUMCIT**2) GO TO 116
    IMIN=99999
    DO 118 I=1,NUMCIT
```

```
      DO 120 J=1,NUMCIT
      IF( IDEL(I,J).NE.0) GO TO 120
      IF(MAT(I,J).LT.IMIN  .AND.  MAT(I,J).GT.0)IMIN=MAT(I,
     *  J)
  120 CONTINUE
  118 CONTINUE
      DO 122 I=1,NUMCIT
      DO 124 J=1,NUMCIT
      IF(IDEL(I,J).EQ.0) MAT(I,J)=MAT(I,J)-IMIN
      IF(IDEL(I,J).EQ.2)MAT(I,J)=MAT(I,J)+IMIN
  124 CONTINUE
  122 CONTINUE
      DO 233 IL=1,NUMCIT
      DO 234 IM=1,NUMCIT
      IF(MAT(IL,IM).GE.0)GO TO 234
      PRINT*," NEGATIVE  NUMBER  IN  MATRIX------>  ABORT"
      PRINT*," GO TO 650"
      GO TO 650
  234 CONTINUE
  233 CONTINUE
      IDSUM=IDSUM+(NUMCIT-ICOUNT)*IMIN
      GO TO 75
   82 ICOUNT=ICOUNT+1
      IX1(ICOUNT)=ILOC
      IX2(ICOUNT)=J
      DO 128 I2=1,NUMCIT
      IDEL(ILOC,I2)=IDEL(ILOC,I2)+1
      IF(MAT(ILOC,I2).EQ.0)MAT(ILOC,I2)=-88888
  128 CONTINUE
      MAT(ILOC,J)=-88888
      GO TO 77
   88 ICOUNT=ICOUNT+1
      IX1(ICOUNT)=I
      IX2(ICOUNT)=ILOC
      DO 130 I2=1,NUMCIT
      IDEL(I2,ILOC)=IDEL(I2,ILOC)+1
      IF(MAT(I2,ILOC).EQ.0)MAT(I2,ILOC)=-88888
  130 CONTINUE
      MAT(I,ILOC)=-88888
      GO TO 77
  116 CONTINUE
      DO 2222 IV=1,NUMCIT
      IF(IX1(IV).NE.IX2(IV)) GO TO 2222
      PRINT*," *****",IX1(IV),IX2(IV)
      GO TO 650
 2222 CONTINUE
      IF(IDSUM.GE.MMIN) GO TO 650
      IHA=3*NUMCIT
      DO 701 IL=1,IHA
      ISLIP(IL)=0
```

```
701 CONTINUE
    ISLIP(1)=1
    ISTOP=1
    IF=1
    IML=99999
720 ICT=1
702 ILOOK=ISLIP(IF)
703 DO 704 IL=1,NUMCIT
    IF2=IL
    IF(IX1(IL).EQ.ILOOK) GO TO 705
704 CONTINUE
705 ISLIP(IF+1)=IX2(IF2)
    ICT=ICT+1
    IF=IF+1
    IF(ISLIP(IF).EQ.ISTOP) GO TO 706
    GO TO 702
706 IF=IF+1
    IF(ICT.EQ.NUMCIT+1) GO TO 790
    IF(ICT.LT.IML) IML=ICT
    ISLIP(IF)=-44444
    DO 707 IL=1,NUMCIT
    DO 708 IM=1,IF
    IF(ISLIP(IM).EQ.IL) GO TO 707
708 CONTINUE
    ISLIP(IF+1)=IL
    IF=IF+1
    ISTOP=IL
    GO TO 720
707 CONTINUE
    IOY=0
    DO 921 IL=1,300
    IF(ISLIP(IL).EQ.-44444) IOY=IOY+1
921 CONTINUE
    IF(IOY.LE.IYART) GO TO 922
    DO 901 IL=1,NUMCIT
    IH=0
    IG=0
904 IG=IG+1
    IF(ISLIP(IG).EQ.-44444) GO TO 902
    IF(ISLIP(IG).EQ.0) GO TO 901
    IH=IH+1
    IBM2(IH)=ISLIP(IG)
    IF(ISLIP(IG).NE.IL) GO TO 904
    IH=0
907 IG=IG+1
    IF(ISLIP(IG).EQ.-44444) GO TO 904
    GO TO 907
902 MINT=99999
    DO 903 IM=1,IH
    IF(MAT(IL,IBM2(IM)).GE.MINT) GO TO 903
```

```
      MINT=MAT(IL,IBM2(IM))
  903 CONTINUE
      DO 905 IM=1,IH
      IF(MAT(IL,IBM2(IM)).EQ.MINT) GO TO 905
      MAT(IL,IBM2(IM))=99999
  905 CONTINUE
      IH=0
      GO TO 904
  901 CONTINUE
  922 CONTINUE
      IK=0
      IP=0
  709 IK=IK+1
      IF(ISLIP(IK).EQ.-44444) GO TO 710
      IP=IP+1
      IBM(IP)=ISLIP(IK)
      GO TO 709
  710 IF(IP.EQ.IML) GO TO 711
      IP=0
      GO TO 709
  711 DO 712 IL=1,NUMCIT
      DO 713 IM=1,NUMCIT
      MAT2(IL,IM)=MAT(IL,IM)
  713 CONTINUE
  712 CONTINUE
      IMM=IML-1
      DO 717 IL=1,IMM
          DO 715 IL2=1,NUMCIT
          DO 716 IM2=1,NUMCIT
          MAT(IL2,IM2)=MAT2(IL2,IM2)
  716 CONTINUE
  715 CONTINUE
      DO 718 IM=1,IMM
      IF(IL.EQ.IM) GO TO 718
      MAT(IBM(IL),IBM(IM))=99999
  718 CONTINUE
      INTC=INTC+1
      WRITE(21,13)ICSUM
      DO 719 IM=1,NUMCIT
      WRITE(21,13)(MAT(IM,IN),IN=1,NUMCIT)
      WRITE(9,13)(MAT(IM,IN),IN=1,NUMCIT)
  719 CONTINUE
  717 CONTINUE
       GO TO 650
  750 CALL RETURN("YES")
      ISKNK=ISKNK+INTC
      ENDFILE 21
      REWIND 21
      CALL RENAME("YES","ZIPP")
      REWIND 5
```

```
      READ(5,*)IDUM
      IF(EOF(5))791,792
  792 GO TO 98849
  790 CONTINUE
      IF(IDSUM.LT.MMIN) MMIN=IDSUM
      GO TO 650
  791 CONTINUE
      MURF(IPPS+1,1)=MURF(IPPS+1,1)+MMIN
      MURF(IPPS+1,2)=MURF(IPPS+1,2)+SECOND(1.)-YYUKK
 7113 CONTINUE
 7114 CONTINUE
      DO 7115 IL=1,IKART
      DO 7115 IM=1,2
      MURF(IL,IM)=MURF(IL,IM)/ITER
 7115 CONTINUE
      PRINT*," ####################"
      DO 7116 IL=1,IKART
      WRITE(11,19)(MURF(IL,IM),IM=1,2)
   19 FORMAT(1X,2F15.3)
 7116 CONTINUE
      STOP
      END
```

```
      SUBROUTINE SPUT2
      DIMENSION MAT(50,50)
      COMMON /LAB7/ISEED,SPR,NUMCIT,IUP
      CALL RANSET(ISEED)
      DO 32 I=1,NUMCIT
      DO 33 K=1,NUMCIT
      MAT(I,K)=IUP*RANF(0.)+1
   33 CONTINUE
   32 CONTINUE
      DO 36 I=1,NUMCIT
      MAT(I,I)=999999
   36 CONTINUE
      REWIND 5
      WRITE(5,14) 0
   14 FORMAT(1X,I5)
      DO 42 I=1,NUMCIT
      WRITE(5,13)(MAT(I,J),J=1,NUMCIT)
   13 FORMAT(1X,10I7/9(4X,9I7/))
   42 CONTINUE
      ENDFILE 5
      REWIND 5
      RETURN
```

```
C       THIS IS THE N-1 NODE BRANCHING HEURISTIC
C   FOR EUCLIDEAN ISEED=ISEED+3
C   FOR NON-EUCLIDEAN ISEED=ISEED+11
      PROGRAM MAIN(YES,INPUT,OUTPUT,MF,GARBAG,EXP,FIN,ZIPP,
     *    UU,PARK30,
     *    TAPE22=UU,TAPE7=PARK30,
     *   TAPE5=YES,TAPE6=EXP,TAPE8=GARBAG,TAPE9=MF,TAPE21=
     *   ZIPP,
     *   TAPE10=INPUT,TAPE11=OUTPUT,TAPE16=FIN)
      COMMON /LAB1/MAT(50,50),IDEL(50,50),IR(50),IC(50),
     *    IX1(50),IX2(50),ISLIP(300)
      COMMON /LAB5/MAT2(50,50),IBM(50)
      COMMON /LAB6/MUCK(300),MUCK2(300)
      COMMON /LAB7/ISEED,SPR,NUMCIT,IUP
      COMMON /LAB8/IUEW
      COMMON /LAB9/MURF(2)
      REAL MURF
      DATA MURF/2*0./
      PRINT*," HOW MANY CITIES"
      PRINT*," WHAT IS SEED ..... UPPER LIMIT"
      PRINT*," NUMBER OF REPS"
      READ(10,*)NUMCIT,ISEED,IUP,ITER
      DO 7114 IQQQ=1,ITER
      PRINT*," AT # ",IQQQ
      ISEED=ISEED+11
      IUEW=ISEED
      YYUKK=SECOND(1.)
      MMIN=99999
      CALL SPUT2
98849 CONTINUE
      INTC=0
      REWIND 8
      REWIND 9
      REWIND 3
      REWIND 21
      REWIND 6
  650 READ(5,*)IDSUM
      IF(EOF(5))750,751
  751 CONTINUE
      READ(5,*)((MAT(I,J),J=1,NUMCIT),I=1,NUMCIT)
      DO 64 I=1,NUMCIT
      MIN=99999
      DO 66 J=1,NUMCIT
      IF(MAT(I,J).LT.MIN  .AND. MAT(I,J).GE.0)MIN=MAT(I,J)
   66 CONTINUE
      IF(MIN.EQ.0) GO TO 64
      DO 68 J=1,NUMCIT
      MAT(I,J)=MAT(I,J)-MIN
   68 CONTINUE
      IR(I)=IR(I)+MIN
```

```
      IDSUM=IDSUM+MIN
   64 CONTINUE
      DO 70 J=1,NUMCIT
      MIN=99999
      DO 72 I=1,NUMCIT
      IF(MAT(I,J).LT.MIN  .AND.  MAT(I,J).GE.0)MIN=MAT(I,J)
   72 CONTINUE
      IF(MIN.EQ.0) GO TO 70
      DO 74 I=1,NUMCIT
      MAT(I,J)=MAT(I,J)-MIN
   74 CONTINUE
      IC(J)=IC(J)+MIN
      IDSUM=IDSUM+MIN
   70 CONTINUE
   75      ICOUNT=0
           DO 81 K=1,NUMCIT
           IX1(K)=0
           IX2(K)=0
   81 CONTINUE
           DO 187 K2=1,NUMCIT
           DO 188 K3=1,NUMCIT
           IDEL(K2,K3)=0
  188 CONTINUE
  187 CONTINUE
   13 FORMAT(1X,9I8/9(4X,9I8/))
   77 NIDUM=0
      DO 78 J=1,NUMCIT
      IDUM=0
      DO 80 I=1,NUMCIT
      IF(MAT(I,J).NE.0) GO TO 80
      IDUM=IDUM+1
      ILOC=I
   80 CONTINUE
      IF(IDUM.EQ.1) GO TO 82
      IF(IDUM.EQ.0)NIDUM=NIDUM+1
   78 CONTINUE
      DO 84 I=1,NUMCIT
      IDUM=0
      DO 86 J=1,NUMCIT
      IF(MAT(I,J).NE.0) GO TO 86
      IDUM=IDUM+1
      ILOC=J
   86 CONTINUE
      IF(IDUM.EQ.1) GO TO 88
      IF(IDUM.EQ.0)NIDUM=NIDUM+1
   84 CONTINUE
      IF(NIDUM.EQ.2*NUMCIT) GO TO 104
      DO 332 M2=1,NUMCIT
      IDUM=0
      DO 334 M4=1,NUMCIT
```

```
      IF(MAT(M2,M4).NE.0) GO TO 334
      IDUM=IDUM+1
      MUCK(IDUM)=M4
  334 CONTINUE
      IF(IDUM.EQ.2) GO TO 336
      GO TO 332
  336 IDUM=0
  337 IDUM=IDUM+1
      IDUM2=0
      DO 340 M8=1,NUMCIT
      IF(MAT(M8,MUCK(IDUM)).NE.0) GO TO 340
      IDUM2=IDUM2+1
      MUCK2(IDUM2)=M8
  340 CONTINUE
      IF(IDUM2.NE.2  .AND.  IDUM.EQ.1) GO TO 337
      IF(IDUM.EQ.2) GO TO 332
      GO TO 342
  332 CONTINUE
      PRINT*," NO TWO ZEROES IN ROW AND COL"
      PRINT*," GO TO 650"
      GO TO 650
  342 ILOC=M2
      J=MUCK(IDUM)
  567 DO 344 M10=1,NUMCIT
      IF(MAT(M10,J).EQ.0) MAT(M10,J)=-88888
  344 CONTINUE
      GO TO 82
  104 CONTINUE
      DO 108 I=1,NUMCIT
      DO 110 J=1,NUMCIT
      IF(MAT(I,J).EQ.-88888)MAT(I,J)=0
  110 CONTINUE
  108 CONTINUE
      ISUM=0
      DO 112 I=1,NUMCIT
      DO 114 J=1,NUMCIT
      ISUM=ISUM+IDEL(I,J)
  114 CONTINUE
  112 CONTINUE
      IF(ISUM.EQ.NUMCIT**2) GO TO 116
      IMIN=99999
      DO 118 I=1,NUMCIT
      DO 120 J=1,NUMCIT
      IF( IDEL(I,J).NE.0) GO TO 120
      IF(MAT(I,J).LT.IMIN  .AND.  MAT(I,J).GT.0)IMIN=MAT(I,
     *  J)
  120 CONTINUE
  118 CONTINUE
      DO 122 I=1,NUMCIT
      DO 124 J=1,NUMCIT
```

```
      IF(IDEL(I,J).EQ.0) MAT(I,J)=MAT(I,J)-IMIN
      IF(IDEL(I,J).EQ.2)MAT(I,J)=MAT(I,J)+IMIN
  124 CONTINUE
  122 CONTINUE
      DO 233 IL=1,NUMCIT
      DO 234 IM=1,NUMCIT
      IF(MAT(IL,IM).GE.0)GO TO 234
      PRINT*," NEGATIVE  NUMBER  IN  MATRIX------>  ABORT"
      PRINT*," GO TO 650"
      GO TO 650
  234 CONTINUE
  233 CONTINUE
      IDSUM=IDSUM+(NUMCIT-ICOUNT)*IMIN
      GO TO 75
   82 ICOUNT=ICOUNT+1
      IX1(ICOUNT)=ILOC
      IX2(ICOUNT)=J
      DO 128 I2=1,NUMCIT
      IDEL(ILOC,I2)=IDEL(ILOC,I2)+1
      IF(MAT(ILOC,I2).EQ.0)MAT(ILOC,I2)=-88888
  128 CONTINUE
      MAT(ILOC,J)=-88888
      GO TO 77
   88 ICOUNT=ICOUNT+1
      IX1(ICOUNT)=I
      IX2(ICOUNT)=ILOC
      DO 130 I2=1,NUMCIT
      IDEL(I2,ILOC)=IDEL(I2,ILOC)+1
      IF(MAT(I2,ILOC).EQ.0)MAT(I2,ILOC)=-88888
  130 CONTINUE
      MAT(I,ILOC)=-88888
      GO TO 77
  116 CONTINUE
      DO 2222 IV=1,NUMCIT
      IF(IX1(IV).NE.IX2(IV)) GO TO 2222
      PRINT*," *****",IX1(IV),IX2(IV)
      STOP
 2222 CONTINUE
      IF(IDSUM.GE.MMIN) GO TO 650
      IHA=3*NUMCIT
      DO 701 IL=1,IHA
      ISLIP(IL)=0
  701 CONTINUE
      ISLIP(1)=1
      ISTOP=1
      IF=1
      IML=99999
  720 ICT=1
  702 ILOCK=ISLIP(IF)
  703 DO 704 IL=1,NUMCIT
```

```
      IF2=IL
      IF(IX1(IL).EQ.ILOOK) GO TO 705
  704 CONTINUE
  705 ISLIP(IF+1)=IX2(IF2)
      ICT=ICT+1
      IF=IF+1
      IF(ISLIP(IF).EQ.ISTOP) GO TO 706
      GO TO 702
  706 IF=IF+1
      IF(ICT.EQ.NUMCIT+1) GO TO 790
      IF(ICT.LT.IML) IML=ICT
      ISLIP(IF)=-44444
      DO 707 IL=1,NUMCIT
      DO 708 IM=1,IF
      IF(ISLIP(IM).EQ.IL) GO TO 707
  708 CONTINUE
      ISLIP(IF+1)=IL
      IF=IF+1
      ISTOP=IL
      GO TO 720
  707 CONTINUE
      IK=0
      IP=0
  709 IK=IK+1
      IF(ISLIP(IK).EQ.-44444) GO TO 710
      IP=IP+1
      IBM(IP)=ISLIP(IK)
      GO TO 709
  710 IF(IP.EQ.IML) GO TO 711
      IP=0
      GO TO 709
  711 DO 712 IL=1,NUMCIT
      DO 713 IM=1,NUMCIT
      MAT2(IL,IM)=MAT(IL,IM)
  713 CONTINUE
  712 CONTINUE
      IMM=IML-1
      DO 730 IL=1,IMM
          DO 731 IL2=1,NUMCIT
          DO 732 IM2=1,NUMCIT
          MAT(IL2,IM2)=MAT2(IL2,IM2)
  732 CONTINUE
  731 CONTINUE
      DO 733 IM=1,IMM
      IF(IL.EQ.IM)GO TO 733
      DO 734 IL3=1,NUMCIT
      MAT(IBM(IM),IL3)=99999
      MAT(IL3,IBM(IM+1))=99999
  734 CONTINUE
      MAT(IBM(IM),IBM(IM))=999999
```

```
      MAT(IBM(IM+1),IBM(IM+1))=999999
      MAT(IBM(IM),IBM(IM+1))=0
  733 CONTINUE
      MAT(IBM(IL),IBM(IL+1))=99999
      INTC=INTC+1
      WRITE(21,13) IDSUM
      DO 719 IM=1,NUMCIT
      WRITE(21,13)(MAT(IM,IN),IN=1,NUMCIT)
  719 CONTINUE
  730 CONTINUE
       GO TO 650
  750 CALL RETURN("YES")
      ENDFILE 21
      REWIND 21
      CALL RENAME("YES","ZIPP")
      REWIND 5
      READ(5,*)IDUM
      IF(EOF(5))791,792
  792 GO TO 98849
  790 CONTINUE
      IF(IDSUM.LT.MMIN) MMIN=IDSUM
      GO TO 650
  791 CONTINUE
      MURF(1)=MURF(1)+MMIN
      MURF(2)=MURF(2)+SECOND(1.)-YYUKK
 7114 CONTINUE
      DO 7115 IL=1,2
      MURF(IL)=MURF(IL)/ITER
 7115 CONTINUE
      PRINT*," ###############"
      WRITE(11,19)(MURF(IL),IL=1,2)
   19 FORMAT(1X,2F15.3)
      STOP
      END




      SUBROUTINE SPUT2
      DIMENSION MAT(50,50)
      COMMON /LAB7/ISEED,SPR,NUMCIT,IUP
      CALL RANSET(ISEED)
      DO 32 I=1,NUMCIT
      DO 33 K=1,NUMCIT
      MAT(I,K)=IUP*RANF(0.)+1
   33 CONTINUE
   32 CONTINUE
      DO 36 I=1,NUMCIT
```

```
      MAT(I,I)=999999
   36 CONTINUE
      REWIND 5
      WRITE(5,14) 0
   14 FORMAT(1X,I5)
      DO 42 I=1,NUMCIT
      WRITE(5,13)(MAT(I,J),J=1,NUMCIT)
   13 FORMAT(1X,10I7/9(4X,9I7/))
   42 CONTINUE
      ENDFILE 5
      REWIND 5
      RETURN
      END
```

REFERENCES

1. Ashour, S., J. Vega and R. G. Parker, "A Heuristic Algorithm for the Traveling Salesman Problem," Transportation Research, Vol. 6, 1971.

2. Bellman, R., "Dynamic Programming Treatment of the Traveling Salesman Problem," J. ACM, Vol. 9, 1962.

3. Bellmore, M. and G. L. Nemhauser, "The Traveling Salesman Problem: A Survey," Operations Research, Vol. 16, 1968.

4. Bellmore, M. and J. C. Malone, "Pathology of Traveling Salesman Subtour Elimination Algorithms," Operations Research, Vol. 19, 1971.

5. Christofides, N., "The Shortest Hamiltonian Chain of a Graph," Journal of SIAM, Vol. 19, 1970.

6. Christofides, N., "Bounds for the Traveling Salesman Problem," Operations Research, Vol. 20, 1972.

7. Christofides, N. and S. Eilon, "Algorithms for Large Scale Traveling Salesman Problems," Operational Research Quarterly, Vol. 23, 1972.

8. Christofides, N., Graph Theory: An Algorithmic Approach, Academic Press, London, 1975.

9. Clark, G. and W. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," Operations Research, Vol. 12, 1964.

10. Dantzig, G. B., D. R. Fulkerson and S. M. Johnson, "Solution of a Large Scale Traveling Problem," Operations Research, Vol. 2, 1954.

11. Dirac, G. A., "Connectivity Theorems for Graphs," Quart. J. Math. Oxford, Ser. (2), Vol. 3, 1952.

12. Eastman, W. L., "Linear Programming with Pattern Constraints," Ph.D. Dissertation, Harvard, 1958.

13. Edmonds, J. "Maximum Matching and a Polyhedron with 0, 1 Vertices," J. Res. Nat. Bureau of Standards, Vol. 696, Nos. 1-2, January-June, 1965.

14. Ford, L. R. and D. R. Fulkerson, Flows in Networks, Princeton University Press, 1962.

15. Gonzalez, R. H., "Solution to the Traveling Salesman Problem by Dynamic Programming on the Hypercube," Tech. Rep. No. 18, O.R. Center, M.I.T., 1962.

16. Held, M., and R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems," SIAM, Vol. 10, 1962.

17. Held, M. and R. M. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees," Operations Research, Vol. 18, 1970.

18. Held, M. and R. M. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees. Part II," Mathematical Programming, Vol. 1, 1971.

19. Holmes, R. A. and R. G. Parker, "A Vehicle Scheduling Procedure Based Upon Savings and a Solution Perturbation Scheme," Operational Research Quarterly, Vol. 27, 1976.

20. Karg, R. L. and G. L. Thompson, "A Heuristic Approach to Solving Traveling Salesman Problems," Management Science, Vol. 10, 1964.

21. Lawler, E. L., "A Solvable Case of the Traveling Salesman Problem," Mathematical Programming, Vol. 1, 1971.

22. Lin, S., "Computer Solution of the Traveling Salesman Problem," Bell System Technical Journal, Vol. 44, 1965.

23. Lin, S. and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," Operations Research, Vol. 21, 1973.

24. Little, J. D. C., K. G. Murty, D. W. Sweeney, and C. Karel, "An Algorithm for the Traveling Salesman Problem," Operations Research, Vol. 11, 1963.

25. Martin, G. T., "An Accelerated Euclidean Algorithm for Integer Linear Programming," Recent Advances in Mathematical Programming, 1963.

26. Miller, C., Tucker, A., and R. Zemlin, "Integer Programming Formulations and Traveling Salesman Problems," Journal of ACM, Vol. 7, 1960.

27. Parker, R. G. and I. J. Perez, "An Application of the Savings Approach to Euclidean Distance Traveling Salesman Problems," Working Paper, Georgia Institute of Technology, 1976.

28. Raymond, T. C., Heuristic Algorithm for the Traveling Salesman Problem," I.B.M. Journal of Research Development, 1969.

29. Reiter, S. and G. Sherman, "Discrete Optimizing," SIAM, Vol. 13, 1965.

30. Roberts, S. M. and B. Flores, "An Engineering Approach to the Traveling Salesman Problem," Management Science, Vol. 13, 1966.

31. Shapiro, D., "Algorithms for the Solution of the Optimal Cost Traveling Salesman Problem," Sc.D. Thesis, Washington University, St. Louis, Missouri, 1966.

32. Svestka, J. and V. Huckfeldt, "Computational Experience with an M-Salesman Traveling Salesman Algorithm," Management Science, Vol. 19, 1973.

33. Syslo, M. M., "A New Solvable Case of the Traveling Salesman Problem," Mathematical Programming, Vol. 4, 1973.

34. Turner, W. C., P. M. Ghare, and L. R. Fourds, "The Transportation Routing Problem - A Survey," AIIE Transactions, Vol. 6, 1975.